UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMATICA
Departamento de Arquitectura de Computadores y Automática

Planificación multinivel eficiente con aprovisionamiento dinámico en grids y clouds

Efficient multilevel scheduling in grids and clouds with dynamic provisioning

MEMORIA PARA OPTAR AL GRADO DE DOCTOR
PRESENTADA POR

Antonio Juan Rubio Montero

Directores

Rafael Mayo García

Francisco Castejón Magaña

Eduardo Huedo Cuesta

Madrid, 2016

ISBN: - 978-84-608-5925-3

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Arquitectura de Computadores y Automática

TESIS DOCTORAL

Planificación Multinivel Eficiente con Aprovisionamiento Dinámico en Grids y Clouds

---

Efficient Multilevel Scheduling in Grids and Clouds with Dynamic Provisioning

Antonio Juan Rubio Montero

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Arquitectura de Computadores y Automática

Planificación Multinivel Eficiente con Aprovisionamiento Dinámico en Grids y Clouds

---

Efficient Multilevel Scheduling in Grids and Clouds with Dynamic Provisioning

Memoria que presenta para optar al título de Doctor en Informática

Antonio Juan Rubio Montero

Dirigida por

Rafael Mayo García
Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas. (CIEMAT).

Francisco Castejón Magaña
Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas. Laboratorio Nacional de Fusión. (LNF-CIEMAT).

Eduardo Huedo Cuesta
Facultad de Informática. Universidad Complutense de Madrid.

Madrid, Spain. 2015

ISBN: 978-84-608-5925-3

A todos los que me quieren, en especial a mi familia.

Acknowledgements

Es difícil nombrar a todas aquellas personas que en algún momento me han ayudado a completar este trabajo, intentaré resumir, aún a costa de olvidarme de alguien importante. En primer lugar quiero agradecer el apoyo y ayuda de mis directores, Paco y Eduardo, pero en especial de Rafa, que ha estado siempre disponible para supervisar toda esta investigación y poder cumplir cualquier plazo de entrega. Sin ellos esta tesis no hubiera sido posible.

Por otro lado, siempre me han respondido a cualquier consulta técnica Javi Fontán, Tino Vázquez, Ismael Marín y José Luis Vázquez Poletti. Además, José Luis ha sido un referente para mí de lo que un investigador tiene que ser, sin ninguna duda. Todos me han ahorrado muchas horas de trabajo y disgustos. Sin embargo, la persona con la que más me alegro haber invertido mi tiempo de trabajo es Manuel Rodríguez Pascual, junto a él muchos logros de esta tesis se han podido llevar a cabo.

También quiero agradecer a Esther Montes, Rosa de Lima Herrera, Jorge Blanco, Ivan Casas y Manolo Giménez que se ocuparan de parte de mis tareas cuando los plazos me obligaban a dedicarme casi exclusivamente a escribir artículos varios. También a mis jefes del CIEMAT, en particular a Fernando Blanco por conceder todas las facilidades posibles para poder realizar una labor de investigación. En este sentido también quiero agradecer a Ignacio Martín Llorente y Rubén Santiago Montero que me ayudaran a iniciar mi carrera investigadora y a todos los colaboradores que han aportado sus conocimientos en diversos artículos.

Finalizo agradeciendo al CIEMAT su apoyo institucional y logístico, al igual que a los proyectos que nombro a continuación:

This work made use of the results produced by the EELA-2, EGEE, GISELA, EGI-InSPIRE and CHAIN-REDS projects, co-founded by the European Commission within its Seventh Framework Programme (Grants: INFRA-2007-223797, RI-222667 , RI-261487 , RI-261323 and RI- 306819). It also has been supported by BETTY (ICT COST Action IC1201). Additionally, part of this work is based on observations obtained with XMM-Newton, an ESA science mission with instruments and contributions directly funded by ESA Member States and the USA (NASA).

About this Document

This thesis is based on the following main publications:

1. A. J. Rubio-Montero, E. Huedo, and R. Mayo-García, "Scheduling multiple virtual environments in cloud federations for distributed calculations," *Future Generation Computer Systems*, p. (Under Review), 2016.

2. A. J. Rubio-Montero, M. A. Rodríguez-Pascual, and R. Mayo-García, "A simple model to exploit reliable algorithms in cloud federations," *Soft Computing*, p. (Under Review), 2016.

3. A. J. Rubio-Montero, E. Huedo, F. Castejón, and R. Mayo-García, "GWpilot: Enabling multi-level scheduling in distributed infrastructures with GridWay and pilot jobs," *Future Generation Computer Systems*, vol. 45, pp. 25–52, April 2015. doi : 10.1016/j.future.2014.10.003

4. A. J. Rubio-Montero, F. Castejón, E. Huedo, and R. Mayo-García, "A novel pilot job approach for improving the execution of distributed codes: application to the study of ordering in collisional transport in fusion plasmas," *Concurrency and Computation: Practice & Experience*, vol. 27, no. 13, pp. 3220–3244, September 2015. doi : 10.1002/cpe.3301

5. A. J. Rubio-Montero, F. Castejón, M. A. Rodríguez-Pascual, E. Montes, and R. Mayo, "Drift Kinetic Equation Solver for Grid (DKEsG)," *IEEE Transactions on Plasma Science*, vol. 38, no. 9, pp. 2093–2101, September 2010. doi : 10.1109/TPS.2010.2055164

6. A. J. Rubio-Montero, E. Huedo, and R. Mayo-García, "User-Guided Provisioning in Federated Clouds for Distributed Calculations," in *Adaptive Resource Management and Scheduling for Cloud Computing (ARMS-CC 2015)*, ser. Lecture Notes in Computer Science, vol. 9438. San Sebastián, Spain: Springer, 20th July 2015, pp. 60–77. doi : 10.1007/978-3-319-28448-4_5

7. A. J. Rubio-Montero, M. A. Rodríguez-Pascual, and R. Mayo-García, "Evaluation of an adaptive framework for resilient Monte Carlo executions," in *30th ACM/SIGAPP Symposium On Applied Computing (SAC'15)*. Salamanca, Spain: ACM New York, 13–17 April 2015, pp. 448–455. doi : 10.1145/2695664.2695890

8. A. J. Rubio-Montero, F. Castejón, E. Huedo, M. Rodríguez-Pascual, and R. Mayo-García, "Performance improvements for the neoclassical transport calculation on Grid by means of pilot jobs," in *Int. Conf. on High Performance Comput. and Simulation (HPCS 2012)*. Madrid, Spain: IEEE CS Press, 2–6 July 2012, pp. 609–615. doi : 10.1109/HPCSim.2012.6266981

9. A. J. Rubio-Montero, L. Flores, F. Castejón, E. Montes, M. Rodríguez-Pascual, and R. Mayo, "Executions of a Drift Kinetic Ecuation solver on Grid," in *18th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2010)*. Pisa, Italy: IEEE CS Press, 17–19 February 2010, pp. 454–459. doi : 10.1109/PDP.2010.40

10. A. J. Rubio-Montero, R. S. Montero, E. Huedo, and I. M. Llorente, "Management of Virtual Machines on Globus Grids using GridWay," in *21st IEEE Int. Parallel and Distributed Processing Symposium (IPDPS 2007)*. Long Beach, USA: IEEE CS Press, 27–30 March 2007, pp. 1–7.doi:10.1109/IPDPS.2007.370548

For the elaboration of this thesis, results from these complementary publications have been also used:

11. M. Rodríguez-Pascual, C. Kanellopoulos, A. J. Rubio-Montero, D. Darriba, O. Prnjat, D. Posada, and R. Mayo-García, "Adapting reproducible research capabilities to resilient distributed calculations," *International Journal of Grid and High Performance Computing*, p. (Accepted), 2016.

12. F. Castejón, A. J. Rubio-Montero, A. López-Fraguas, E. Ascasíbar, and R. Mayo-García, "Neoclassical transport and iota scaling in the TJ-II stellarator," *Fusion Science and Technology*, p. (Accepted), 2016.

13. M. Rodríguez-Pascual, A. Gómez, R. Mayo-García, D. P. de Lara, E. M. González, A. J. Rubio-Montero, and J. L. Vicent, "Superconducting Vortex Lattice Configurations on Periodic Potentials: Simulation and Experiment," *Superconductivity and Novell Magnetism*, vol. 25, no. 7, pp. 2127–2130, October 2012. doi : 10.1007/s10948-012-1636-8

14. M. Rodríguez-Pascual, J. Guasp, F. Castejón, A. J. Rubio-Montero, I. M. Llorente, and R. Mayo, "Improvements on the Fusion Code FAFNER2," *IEEE Transactions on Plasma Science*, vol. 38, no. 9, pp. 2102–2110, September 2010. doi : 10.1109/TPS.2010.2057450

15. M. Rodríguez-Pascual, A. J. Rubio-Montero, R. Mayo-García, C. Kanellopoulos, O. Prnjat, D. Darriba, and D. Posada, "A fault tolerant workflow for reproducible research," in *Annual Global Online Conference on Information and Computer Technology (GOCICT 2014)*. Louisville, Kentucky, USA: IEEE CS Press, 3–5 December 2014, pp. 70–75. doi : 10.1109/GOCICT.2014.10

16. R. Isea, E. Montes, A. J. Rubio-Montero, and R. Mayo, *State-of-Art with PhyloGrid: Grid Computing Phylogenetic Studies on the EELA-2 Project Infrastructure*, in *Grid Computing: Towards a Global Interconnected Infrastructure*, ser. Computer Communications and Networks. Springer London / Heidelberg New York, 2011, pp. 277–291. doi : 10.1007/978-0-85729-676-4_11

17. M. Rodríguez-Pascual, A. J. Rubio-Montero, R. Mayo, A. Bustos, F. Castejón, and I. Llorente, "More Efficient Executions of Monte Carlo Fusion Codes by Means of Montera: The ISDEP Use Case," in *19th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2011)*. Ayia Napa, Cyprus: IEEE CS Press, 9–11 February 2011, pp. 380–384.doi:10.1109/PDP.2011.46

18. M. A. Rodríguez-Pascual, J. Guasp, F. Castejón, A. J. Rubio-Montero, I. M. Llorente, and R. Mayo, "A Grid version of the Fusion code FAFNER," in *18th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2010)*. Pisa, Italy: IEEE CS Press, 17–19 February 2010, pp. 449–453. doi : 10.1109/PDP.2010.37

19. R. Isea, E. Montes, A. J. Rubio-Montero, J. D. Rosales, M. A. Rodríguez-Pascual, and R. Mayo, "Characterization of antigenetic serotypes from the dengue virus in Venezuela by means of Grid Computing," in *Healthgrid Applications and core Technologies. Proceedings of HealthGrid 2010*, ser. Studies in Health Technology and Informatics, vol. 159. Paris, France: IOS Press, 28–30 June 2010, pp. 234–238. doi : 10.3233/978-1-60750-583-9-234

20. M. Rodríguez-Pascual, F. Castejón, A. J. Rubio-Montero, R. Mayo, and I. M. Llorente, "FAFNER2: A comparison between the Grid and the MPI versions of the code," in *Int. Conf. on High Performance Comput. and Simulation (HPCS 2010)*. Caen, France: IEEE CS Press, 28 June–2 July 2010, pp. 78–84. doi : 10.1109/HPCS.2010.5547146

21. M. Rodíguez-Pascual, D. P. de Lara, E. M. González, A. Gómez, A. J. Rubio-Montero, R. Mayo, and J. Vicent, "Grid computing simulation of superconducting vortex lattice in superconducting magnetic nanostructures," in *Proceedings of the 4th Iberian Grid Infrastructure Conference*, vol. 4. Braga, Portugal: NETBIBLO S.L. (Sta. Cristina, La Coruña, Spain), 24–27 May 2010, pp. 97–109. ISBN 978-84-9745-549-7

22. R. Isea, E. Montes, A. J. Rubio-Montero, and R. Mayo, "Computational Challenges on Grid Computing for Workflows Applied to Phylogeny," in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living. 10th Int. Work-Conference on Artificial Neural Networks (IWANN 2009)*, ser. Lecture Notes in Computer Science, vol. 5518. Salamanca, Spain: Springer-Verlag, 10–12 June 2009, pp. 1130–1138. doi : 10.1007/978-3-642-02481-8_171

23. A. J. Rubio-Montero, P. Arce, J. I. Lagares, Y. P. Ivanov, D. A. Burbano, G. Díaz, and R. Mayo, "Performance Tests of GAMOS Software on EELA-2 Infrastructure," in *Proceedings of the Second EELA-2 Conference*. Choroní, Venezuela: Editorial CIEMAT (Madrid, Spain), 25–27 November 2009, pp. 379–385. ISBN 978-84-7834-627-1

24. R. Isea, J. Chaves, E. Montes, A. J. Rubio-Montero, and R. Mayo, "The evolution of HPV by means of a phylogenetic study," in *Healthgrid Research, Innovation and Business Case. Proceedings of HealthGrid 2009*, ser. Studies in Health Technology and Informatics, vol. 147. Berlin, Germany: IOS Press, 29 June–1 July 2009, pp. 245–250. doi : 10.3233/978-1-60750-027-8-245

Abstract

The consolidation of large Distributed Computing infrastructures has resulted in a High-Throughput Computing platform that is ready for high loads, whose best proponents are the current grid federations. On the other hand, Cloud Computing promises to be more flexible, usable, available and simple than Grid Computing, covering also much more computational needs than the ones required to carry out distributed calculations. In any case, because of the dynamism and heterogeneity that are present in grids and clouds, calculating the best match between computational tasks and resources in an effectively characterised infrastructure is, by definition, an NP-complete problem, and only sub-optimal solutions (schedules) can be found for these environments. Nevertheless, the characterisation of the resources of both kinds of infrastructures is far from being achieved. The available information systems do not provide accurate data about the status of the resources that can allow the advanced scheduling required by the different needs of distributed applications. The issue was not solved during the last decade for grids and the cloud infrastructures recently established have the same problem. In this framework, brokers only can improve the throughput of very long calculations, but do not provide estimations of their duration. Complex scheduling was traditionally tackled by other tools such as workflow managers, self-schedulers and the production management systems of certain research communities. Nevertheless, the low performance achieved by these early-binding methods is noticeable. Moreover, the diversity of cloud providers and mainly, their lack of standardised programming interfaces and brokering tools to distribute the workload, hinder the massive portability of legacy applications to cloud environments.

Unlike the aforementioned early-binding methods, where the workload is scheduled in resources before they have been effectively assigned, the relatively recent pilot job technique (or late-binding model) is being introduced into grids and clouds to overcome the current limitations of middleware. This approach accomplishes computational tasks in a more flexible, stable and reliable way while reducing overheads. Furthermore, it constitutes a powerful scheduling layer that can be used to solve the aforementioned characterisation problems and can be combined with traditional mechanisms to achieve the required performance levels. In this sense, this new scheduling overlay is logically placed between the ones provided by the applications and the provisioning tools, building a Multilevel Scheduling architecture. Nevertheless and even when they have provided a clear improvement in the execution of distributed calculations, the current systems based on pilot jobs are not exploiting all the advantages that this technique could afford in grids and clouds, or they lack compatibility or adaptability. In particular, these tools are not flexible enough to support the characterisation needed, to customise the behaviour of the scheduling layers or to dynamically configure the provisioned grid and cloud workspace. Moreover, some are

unable to run legacy applications based on accepted standards. These issues prevent their application to other fields or codes different to the ones for which they were developed.

The research performed in this thesis overcomes the limitations of current pilot systems to fully exploit the Multilevel Scheduling in grid and cloud environments. This study presents the design of a different general-purpose pilot framework, GWpilot, and a collection of related methodologies and technologies that allow users, developers and institutional administrators to easily incorporate their legacy applications, scheduling frameworks and policies into the Multilevel model, achieving the performance required for the calculations and preserving the compatibility and security among infrastructures. Moreover, the system provides individual users or institutions with a more easy-to-use, easy-to-install, scalable, extendable, flexible and adjustable pilot framework than the current ones. Furthermore, the system also differentiates from other approaches in its decentralisation, its middleware independence, its support of brokering specific workspaces in cloud, the post-configuration of the provisioned resources by users and the efficient accomplishment of short tasks, among other features. All those are performed preserving the fair-share and compatibility with diverse infrastructures. To demonstrate these achievements, the new framework has been implemented and tested with different legacy applications and scheduling policies, performing meaningful calculations on cloud and grid infrastructures in production. Consequently, the new pilot system is available to be profited by the scientific and industrial communities as well as additional contributions to other research areas have been achieved. In this sense, calculations devoted to Chemical Physics, Evolutionary Biology, High Energy Physics, Matter Interactions and Solid State Physics have been performed with GWpilot on grids and clouds, due to their different computational behaviour. Interesting new scientific results have been obtained too, in especial for Nuclear Fusion.

Furthermore, the development of this new pilot job system represents a step forward in the use of large distributed computing infrastructures, because it goes beyond establishing simple network overlays to overcome the waiting times in remote grid queues, making use of cloud resources or improving reliability in task production. It properly tackles the characterisation problem in current infrastructures, allowing users to arbitrarily incorporate a customised monitoring of resources and their running applications into the system. Any user can easily take advantage of this feature to perform a specialised scheduling of his application workload without the need of modifying any code in the pilot system. Users can also automatically guide the provisioning among different grid and cloud providers without the need of explicitly indicating one resource or manually submitting pilots. Moreover, an accurate mathematical model on task turnaround has been formulated to allow building complex scheduling. Subsequently, all of these features can also benefit skilled developers or administrators. Even self-schedulers or workflow managers can be easily adapted to establish an upper scheduling layer that will take into account the real characteristics of resources. This last achievement has been demonstrated by stacking a generic self-scheduler on the system, obtaining an improved performance. These goals were not really registered before and constitute the main intellectual contribution of this thesis.

Summing-up, the Multilevel Scheduling approach presented in this thesis actually allows individual researchers and communities that rely on high-throughput calculations to efficiently profit of the large volume of distributed and heterogeneous resources

available in grids and clouds. This conclusion is supported by the extensive experiments performed on real infrastructures, which have demonstrated its capacity to easily incorporate legacy distributed applications, to customise the characterisation of resources, to personalise the different scheduling layers and to stack third-party schedulers. Therefore, the impact of this research on Computer Science as well as on any other field is guaranteed by the practicality, extensibility, compatibility, performance and multiple possibilities of scheduling that the proposed solution allows.

Resumen (Spanish)

La consolidación de las grandes infraestructuras para la Computación Distribuida ha resultado en una plataforma de Computación de Alta Productividad que está lista para grandes cargas de trabajo. Los mejores exponentes de este proceso son las federaciones grid actuales. Por otro lado, la Computación Cloud promete ser más flexible, utilizable, disponible y simple que la Computación Grid, cubriendo además muchas más necesidades computacionales que las requeridas para llevar a cabo cálculos distribuidos. En cualquier caso, debido al dinamismo y la heterogeneidad presente en grids y clouds, encontrar la asignación ideal de las tareas computacionales en los recursos disponibles es, por definición un problema NP-completo, y sólo se pueden encontrar soluciones subóptimas para estos entornos. Sin embargo, la caracterización de estos recursos en ambos tipos de infraestructuras es deficitaria. Los sistemas de información disponibles no proporcionan datos fiables sobre el estado de los recursos, lo cual no permite la planificación avanzada que necesitan los diferentes tipos de aplicaciones distribuidas. Durante la última década esta cuestión no ha sido resuelta para la Computación Grid y las infraestructuras cloud establecidas recientemente presentan el mismo problema. En este marco, los planificadores (*brokers*) sólo pueden mejorar la productividad de las ejecuciones largas, pero no proporcionan ninguna estimación de su duración. La planificación compleja ha sido abordada tradicionalmente por otras herramientas como los gestores de flujos de trabajo, los auto-planificadores o los sistemas de gestión de producción pertenecientes a ciertas comunidades de investigación. Sin embargo, el bajo rendimiento obtenido con estos mecanismos de asignación anticipada (*early-binding*) es notorio. Además, la diversidad en los proveedores cloud, la falta de soporte de herramientas de planificación y de interfaces de programación estandarizadas para distribuir la carga de trabajo, dificultan la portabilidad masiva de aplicaciones legadas a los entornos cloud.

En contraste con los métodos mencionados, por los cuáles la carga de trabajo es planificada en los recursos antes que éstos sean efectivamente apropiados, la técnica relativamente reciente de los trabajos piloto (*late-binding*) está siendo introducida en grids y clouds para superar las limitaciones del middleware. Esta aproximación permite completar las tareas computacionales de los usuarios de un modo más flexible, estable y robusto, mientras la sobrecarga se reduce. Además, constituye una potente capa de planificación que puede ser usada para resolver los problemas de caracterización mencionados y ser combinada también con los mecanismos tradicionales para alcanzar los niveles de rendimiento requeridos. En este sentido, esta nueva capa está ubicada entre la que proporcionan las aplicaciones y las herramientas de aprovisionamiento, construyendo una arquitectura de Planificación Multinivel. Aunque los actuales sistemas basados en trabajos piloto han permitido una clara mejora en la ejecución de los cálculos distribuidos, no están explotando todas las ventajas que es-

ta técnica puede ofrecer, o bien son poco compatibles y adaptables. En particular, estas herramientas no son suficientemente flexibles para soportar la caracterización necesaria, personalizar el comportamiento de las capas de planificación o configurar dinámicamente los espacios virtuales de trabajo que han sido aprovisionados en grids y clouds. Incluso algunos son incapaces de ejecutar aplicaciones legadas basadas en estándares aceptados. Estas cuestiones impiden su utilización en otros campos o con códigos diferentes para los que fueron diseñados.

La investigación realizada en esta tesis supera las limitaciones de los sistemas de trabajos piloto actuales para explotar completamente la Planificación Multinivel en entornos grid y cloud. Este estudio presenta el diseño de un nuevo sistema de trabajos piloto de propósito general, GWpilot, junto con una colección de metodologías y tecnologías que permiten a los usuarios, desarrolladores y administradores institucionales incorporar fácilmente sus aplicaciones legadas, sus sistemas de planificación y sus políticas en el modelo Multinivel, alcanzando el rendimiento requerido mientras se preserva la compatibilidad y seguridad en las diferentes infraestructuras. Además, el sistema es mucho más fácil de usar, instalar, apilar, extender y ajustar que las herramientas existentes gracias a su flexibilidad. El nuevo sistema se diferencia también en su descentralización, su independencia del middleware, la planificación bajo demanda de diferentes máquinas virtuales en clouds, la configuración posterior por parte de los usuarios de los espacios virtuales aprovisionados, la ejecución eficiente de tareas cortas, todo ello preservando la compartición equitativa de recursos y la compatibilidad con las diversas infraestructuras. Para demostrar estos logros, el nuevo sistema ha sido implementado y probado con diferentes aplicaciones legadas y políticas de planificación, llevando a cabo cálculos significativos sobre infraestructuras grid y cloud en producción. Consecuentemente, no sólo el nuevo sistema está disponible para ser aprovechado por las comunidades científicas e industriales, también se han obtenido resultados en otros campos de investigación. En este sentido, cálculos dedicados a la Fisicoquímica, Biología Evolutiva, Física de las Altas Energías, Interacciones con la Materia y Física del Estado Sólido se han llevado a cabo con GWpilot en grids y clouds debido a su diferente comportamiento computacional, pero además se han obtenido interesantes nuevos resultados, en especial para la Fusión Nuclear.

El desarrollo de este nuevo sistema de trabajos piloto representa un paso adelante en el aprovechamiento de las grandes infraestructuras computacionales distribuidas, ya que va más allá de establecer simplemente una envoltura para eliminar los tiempos de espera en las colas grid remotas, permitir el uso de recursos cloud o mejorar la robustez en la producción de tareas. Aborda apropiadamente el problema de la caracterización en las infraestructuras actuales, permitiendo a los usuarios incorporar arbitrariamente una monitorización personalizada de los recursos y sus aplicaciones ejecutadas en el sistema. Cualquier usuario puede fácilmente aprovechar esta funcionalidad para realizar una planificación especializada de la carga de trabajo de su aplicación, sin tener que modificar el código del sistema. Además pueden guiar automáticamente el aprovisionamiento entre los diferentes proveedores grid y cloud sin tener que indicar explícitamente qué proveedor usar o enviar manualmente los trabajos piloto. Así mismo, un modelo matemático preciso de la duración de las tareas ha sido formulado para poder elaborar planificaciones complejas. Por lo tanto, todas estas funcionalidades distintivas pueden beneficiar también a los desarrolladores expertos o a los administradores. Incluso los auto-planificadores y los gestores de flujos de trabajos pueden ser fácilmente adaptados para establecer una capa superior de planificación que tome en consideración las características reales de los recursos. Este

logro final ha sido demostrado acoplando un auto-planificador genérico al sistema, obteniendo un rendimiento mejorado. Estos logros no habían sido registrados antes y constituyen la principal aportación intelectual de esta tesis.

En resumen, la aproximación a la Planificación Multinivel presentada en esta tesis realmente permite, tanto a investigadores individuales como a comunidades de usuarios que dependen de los cálculos de alta productividad, beneficiarse de forma eficiente del gran volumen de recursos distribuidos y heterogéneos disponibles en grids y clouds. Esta conclusión está fundamentada por los extensos experimentos realizados sobre infraestructuras reales, los cuáles han demostrado su capacidad para incorporar aplicaciones legadas, personalizar la caracterización de los recursos y las diferentes capas de planificación, así como para acoplar software planificador de terceros. Por lo tanto, el impacto de esta investigación en los diferentes campos de la Informática como en otras áreas está garantizada por la practicidad, extensibilidad, compatibilidad, rendimiento y las múltiples posibilidades de planificación avanzada que la solución propuesta permite.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AAI authentication and authorization infrastructure.

AGWL Abstract Grid Workflow Language.

ALICE A Large Ion Collider Experiment.

AliEn ALICE Environment.

AMWAT AppLeS Master Worker Application Template.

APEL Accounting Processor for Event Logs.

API application programming interface.

AppLeS Application Level Scheduling framework.

ARC Advanced Resource Connector.

ATLAS A Toroidal LHC ApparatuS.

AWS Amazon Web Services.

BDII Berkeley Database Information Index.

BOINC Berkeley Open Infrastructure for Network Computing.

BoT bag of tasks.

CA certification authority.

CCF current calibration files.

CERN European Organization for Nuclear Research (Conseil Européen pour la Recherche Nucléaire).

ClassAds Classified Advertisements language.

CLI command line interface.

CMS Compact Muon Solenoid.

CMUL normalised collisionality.

COMPSs COMP Superscalar.

CORBA Common Object Request Broker Architecture.

CREAM Computing Resource Execution And Management.

DAG directed acyclic graph.

DCI Distributed Computing infrastructure.

DEISA Distributed European Infrastructure for Supercomputing Applications.

DHCP Dynamic Host Configuration Protocol.

DIANE Distributed Analysis Environment.

DIRAC Distributed Infrastructure with Remote Agent Control.

DiVoS Superconducting Vortex Dynamics (Dínamica de Vórtices Superconductores).

DKES Drift Kinetic Equation solver.

DKEsG Drift Kinetic Equation solver for Grids.

DKEsG-Mono DKEsG monoenergetic module.

DKEsG-Neo DKEsG Neoclassical module.

DRMAA Distributed Resource Management Application API.

DyTSS Dynamic Trapezoid Self-Scheduling.

EDG European DataGrid Project.

EDGeS Enabling Desktop Grids for e-Science.

EELA-2 E-science grid facility for Europe and Latin America.

EFIELD normalised electrical field.

EGEE Enabling Grids for E-sciencE.

EGI European Grid Infrastructure.

EM Execution Manager.

EMI European Middleware Initiative.

EUGridPMA European Policy Management Authority for Grid Authentication.

Falkon Fast and Light-weight tasK executiON framework.

FCFS first come first serve.

FedCloud EGI Federated Cloud.

FRFS fit resource first serve.

GAMOS GEANT4-based Architecture for Medicine-Oriented Simulations.

GARUDA Global Access to Resources Using Distributed Architecture.

GASS Globus Access to Secondary Storage.

GENIUS Grid Enabled Web Environment for Site Independent User Job Submission.

GFDL Grid Workflow Description Language.

GGUS Global Grid User Support.

GISELA Grid Initiatives for e-Science Virtual Communities in Europe and Latin America.

GLUE Grid Laboratory Uniform Environment.

GPFS General Parallel File System.

GPU graphics processing unit.

GRAM Grid Resource Allocation Manager.

GridFTP Grid File Transfer Protocol.

GridSAM Grid job Submission And Monitoring web service.

GS grid scheduler.

GSFL Grid Services Flow Language.

GT Globus Toolkit.

gUSE grid and cloud User Support Environment.

GW PiF GWpilot Factory.

GW PiS GWpilot Server.

GWcloud ED GWcloud Execution Driver.

GWcloud ID GWcloud Information Driver.

GWEL Grid Workflow Execution Language.

GWorkflowDL Grid Workflow Description Language.

HEP High Energy Physics.

HPC High Performance Computing.

HTC High Throughput Computing.

HTTP Hypertext Transfer Protocol.

IaaS infrastructure-as-a-service.

IM Information Manager.

IP Internet Protocol.

IS information systems.

ISDEP Stochastic Differential Equations for Plasmas.

IWIR Interoperable Workflow Intermediate Representation.

JDL Job Description Language.

JSDL Job Submission Description Language.

K-Wf Knowledge-based Workflow System for Grid Applications.

LAN local area network.

LCG LHC Computing Grid.

LDAP Lightweight Directory Access Protocol.

LHC Large Hadron Collider.

LHCb Large Hadron Collider beauty.

LJF longest job first.

LRMS local resource management system.

M/W Master/Worker.

MA message accumulator.

MAD middleware access driver.

MC Monte Carlo.

MPI Message Passing Interface.

MUPJ multi-user pilot jobs.

NAT network address translation.

NC Neoclassical.

NFS Network File System.

NIS Network Information Service.

OASIS Organization for the Advancement of Structured Information Standards.

OCCI Open Cloud Computing Interface.

ODF observation data files.

OGF Open Grid Forum.

OGSA-BES Open Grid Services Architecture - Basic Execution Service.

OpenMP Open Multi-Processing.

OS Operating System.

OSG Open Science Grid.

P-GRADE Parallel Grid Run-time and Application Development Environment.

PaaS platform-as-a-service.

PanDA Production and Distributed Analysis.

PBS Portable Batch System.

PC personal computer.

PiF pilot factory.

PiS pilot server.

PMES Programming Model Enactment Service.

PMS production manager system.

PRACE Partnership for Advanced Computing in Europe.

PVM Parallel Virtual Machine.

QoS quality of service.

RB resource broker.

REST Representational State Transfer.

RMI Java Remote Method Invocation.

RPC remote procedure call.

RSH Remote shell.

SaaS software-as-a-service.

SAGA Simple API for Grid Applications.

SAM Service Availability Monitoring.

SAS XMM-Newton Science Analysis System.

ServiceSs Service Superscalar.

SG scientific gateway.

SHIWA SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs.

SJF shortest job first.

SLA service level agreement.

SNMP Simple Network Mangement Protocol.

SOA service oriented architecture.

SOAP Simple Object Access Protocol.

SRM Storage Resource Manager.

SSH Secure Shell.

SSL Secure Socket Layer.

SWFL Service Workflow Language.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UMD Unified Middleware Distribution.

UML Unified Modelling Language.

UNICORE Uniform Interface to Computing Resources.

URI Uniform Resource Identifier.

UTQ user task queue.

VIM virtual infrastructure manager.

VM virtual machine.

VO virtual organization.

VOMS Virtual Organization Membership Service.

VPN virtual private network.

VWS Virtual Workspace Service.

WAN wide area network.

WfE workflow engine.

WfM workflow manager.

WMS gLite/UMD Workload Management System.

WS web service.

WS-BPEL Business Process Execution Language for WS.

WSDL Web Service Description Language.

WSRF Web Service Resource Framework.

XPDL XML Process Definition Language.

XSEDE eXtreme Science and Engineering Environment.

YAWL Yet Another Workflow Language.

Chapter 1

Overview

1.1. Motivation

The consolidation of Distributed Computing infrastructures (DCIs) is a long process involving continuous changes, standardisation processes, upgrades of middleware tools, implementation of new capabilities, dissemination of their use, adaptation of applications, etc. In any case, this effort has enabled the integration of a large pool of resources around the world, resulting in a High-Throughput Computing (HTC) platform that is ready for high loads whose best proponents are the current grid federations. Moreover, with the advent of infrastructure-as-a-service (IaaS) clouds (and even, the Volunteer Computing or the services oriented to Big Data), the number of resources and capabilities have exponentially increased, which opens the door to face new research challenges. However, despite recent advances, the work is not yet complete, particularly for job scheduling. Users, developers and site administrators continuously experience poor performance, complexity, and resource underutilisation.

Because of the dynamism and heterogeneity that are present in the majority of DCIs, especially those based on Grid [1] and Cloud [2] Computing, calculating the best match between computational tasks and resources in an effectively characterised infrastructure is, by definition, an NP-complete problem [3], and only sub-optimal solutions (schedules) can be found for these environments. Nonetheless, the persistent problems on DCIs have avoided the deployment of those complex algorithms in production, even though they have been well-tested on simulators and in controlled environments. Grid and cloud information systems (IS) [4] (usually based on GLUE [5]), which are commonly misconfigured, do not provide an effective or reliable characterisation of the offered resources.

The basic information is not provided for scheduling. IS do not currently show any information about the bandwidth or latency, neither the quotas established in cloud providers. In the case of grid infrastructures, for example, the shared policy of the remote queues and the average waiting time are unknown to users. As a consequence, no valuable functionality devoted to advanced reservation of resources has been implemented so far. Therefore, current middleware does not allow the knowledge of the type of resources that will be effectively assigned *a priori*. A lack of tools for generic user application profiling is an added difficulty. Another problem is the continuous overload of centralised services and resources, which increases the response lag and queue times, resulting in poor turnaround of individual jobs.

According to the performance aspect considered [6], the scheduling problem can be oriented to increase the throughput and the resource utilisation of the infrastructure, to reduce the application makespan or to improve the user share, the prioritisation and the data allocation. Algorithms for these policies are widely utilised by local resource management systems (LRMS) to exploit the clusters and constellations belonging to the same institution. Because their computational environments experience few changes, their schedules are based on static descriptions of resources. On the other hand, any developer (or skilled user) of any HTC application should take account of localisation and performance of the available resources to reduce the final makespan of his calculation, the process of which requires from some abstraction. This is so because these specialised users need effective mechanisms to access resources in a way that they could choice between a unified, abstracted view of the infrastructure as a whole, and the opportunity to target specific providers for their needs. For this purpose, users should rely on *brokers* in the first instance. Resource brokers (RBs) are cognisant of grid and cloud dynamism, and they constitute the first mechanism to reduce DCI complexity, thus providing automated and unattended access to all the resources.

Diverse implementations of grid schedulers (GSs) have been proposed, but a few GSs have survived because of their adaptation to the grid complexity, scalability or ease of use. Currently, the GSs based on Condor [7] are the predominant ones in the largest grid infrastructures in its two versions: Condor-G [8] on Open Science Grid (OSG[1]) and gLite/UMD WMS [9] on the European Grid Infrastructure (EGI[2]). GridWay [10] is a potential alternative. However, neither Condor nor GridWay have yet addressed consistent methods to overcome the resource characterisation problem, despite the implementation of behaviour models [11, 12, 13]. The consequences are especially evident when executing short jobs, in which the middleware overhead represents an important percentage of the consumed time. As a result, GSs are devoted to improve the throughput of long jobs from an organisational point of view.

In current IaaS clouds, basic brokering capabilities are still far from being provided for HTC. Cloud Computing promises to be more flexible, usable, available, and simple than Grid Computing, covering also much more computational needs than the ones required to carry out distributed calculations. Nevertheless, the diversity of IaaS cloud providers makes difficult to design a general-purpose broker and the available ones are devoted to service consolidation. Thus, the lack of standardised programming interfaces and brokering tools to distribute the workload hinder the massive portability of legacy applications to cloud environments.

To overcome the lack of reliable information provided from the IS, many application-oriented frameworks have assumed some of its roles and other functionalities that initially correspond to the RBs to improve the quality and quantity of the supplied resources, such as statistical accounting or remote profiling. In this sense, GSs are commonly used as a tool for resource provisioning or dispatching [14]. Moreover, because those frameworks select the resources, they are able to re-implement basic functionalities from the RBs. Systems included in this category are some workflow managers [15], production manager systems (PMS) [16, 17, 18, 19], and self-schedulers [20, 21, 22], which promise a better performance based on a deeper knowledge of the computational needs of a specific application. This is an expensive approach that increases the intrinsic difficulty in testing innovative algorithms in a real environment. Another

[1] http://www.opensciencegrid.org
[2] http://www.egi.eu

issue is the specialisation of such specific types of applications. As a consequence, the performance gain over those applications provided by a general-purpose GS is usually measured in controlled environments, such as simulators, small laboratories or infrastructures with only a few production nodes. Thus, although they show interesting results, their improvements cannot usually be extrapolated to real infrastructures or to different application types with the same good performance.

Unlike the aforementioned early-binding methods, where the workload is scheduled to resources before they have been effectively assigned, the relatively recent pilot job technique (or late-binding model) is being introduced in DCIs to overcome the current limitations of middleware. This approach accomplishes computational user tasks in a more flexible, stable and reliable way while reducing overhead. Moreover, it constitutes a powerful scheduling layer that can be combined with traditional mechanisms to achieve the required performance levels. In this sense, this new scheduling overlay is logically placed between the ones provided by the applications and the provisioning tools, and can be used to solve the aforementioned characterisation problems. Thus, pilot frameworks build Multilevel Scheduling systems that are composed by the Application (or User) layer, the introduced Task Scheduling layer, and the Resource Provisioning layer.

Nevertheless and even when they have provided a clear improvement in the execution of distributed calculations, the current systems based on pilot jobs [17, 19, 23, 24, 25, 26, 27, 28, 29, 30, 31] are not exploiting all the advantages that the technique could afford in grids and clouds, or they lack compatibility or adaptability. In particular, these tools are not flexible enough to support the characterisation needed, to customise the behaviour of the scheduling layers or to dynamically configure the provisioned grid and cloud workspace. Moreover, although current pilot systems are accomplishing great results in the area for which they have been implemented and designed, their limitations prevent their deployment and application to other fields or codes.

Therefore, the aim of this thesis is overcome the limitations of current pilot frameworks to fully exploit the Multilevel Scheduling in grid and cloud environments. This includes allowing users, developers and institutional administrators to easily incorporate their legacy applications, frameworks and scheduling policies into the Multilevel architecture, achieving the performance required for these calculations and preserving the compatibility and security among infrastructures. For this purpose it is necessary an extensive study of the related work and the clarification of concepts to keep the advantages of previous works and to found new suitable approaches.

The main intellectual contribution of the research presented in this thesis is the collection of methodologies and technologies that result in the design of a new pilot framework, which accomplishes these stated requirements. To demonstrate these achievements, the new framework has been implemented and tested with different legacy applications and scheduling systems, performing meaningful calculations on cloud and grid infrastructures in production. Consequently, the new pilot system is available to be profited by the scientific and industrial communities as well as additional contributions to other research areas have been achieved.

Furthermore, the development of this system represents a step forward in the use of DCIs because it goes beyond establishing simple network overlays to overcome the waiting times in remote grid queues, making use of cloud resources or improving reliability in task production. It properly tackles the characterisation problem in current infrastructures, allowing users to arbitrarily incorporate customised monitoring of

resources and their running applications into the system. Any user can easily take advantage of this feature to perform a specialised scheduling of his application workload without the need of modifying any code in the pilot system. Users can also automatically guide the Provisioning among different grid and cloud providers without the need of explicitly indicating one resource or manually submitting pilots. Moreover, all of these features can also benefit skilled developers or administrators to build complex scheduling policies such as the ones in [6, 32]. Additionally, self-schedulers or workflow managers can be easily adapted to establish an upper scheduling layer that will take into account the real characteristics of resources, even when they were federated.

Therefore, the global contribution of this thesis to the Distributed Computing field is remarkable, allowing the efficient profiting of large volume of distributed heterogeneous resources by individual researchers and communities that rely on any kind of HTC calculation. Thus, the impact on Computer Science as well as on any other field is guaranteed by the practicality, extensibility and the multiple possibilities of scheduling that the proposed solution allows.

1.2. Objective, methodology and structure

The objective of this thesis is to found mechanisms to fully profit from the advantages that the Multilevel Scheduling can provide in grid and cloud environments. This is, the capacity to:

- perform several specialised scheduling at every level adapted to the computational and organisational needs;
- make the most from resources belonging to different DCIs;
- create personalised virtual environments on-demand and maintain them during the calculations.

Moreover, the vision under which this research has been carried out results in that, the positive experience and the expensive work performed in large and collaborative grid federations must be preserved as long as possible. Thus the research must assure the:

- support of legacy applications and tools with scheduling capacities;
- the security and compatibility among DCIs;
- the fair-share in a competitive environment.

These items were not previously achieved together by any pilot system. Therefore, the main result of this thesis is to offer a new general-purpose pilot system that accomplishes these capacities for the scientific and industrial community. For this purpose, the following main points must be performed:

a) an extensive study on the advances previously carried out and the weaknesses found, which allows determining the suitable techniques to be applied as well as defining a detailed list of design requirements for a new pilot system;

b) an evaluation and improvement of the existent early-binding techniques, or even the creation of new ones if needed, to be posteriorly profited following a late-binding approach.

c) the design of the new system and its consequently validation, customising the scheduling layers, incorporating third-party schedulers and stressing the framework with real executions on real grid and cloud infrastructures in production.

d) the demonstration of the final suitability of the solution for diverse scientific fields, obtaining valuable results in their scope.

With the achievement of these items, the intellectual contribution of this thesis to the Computer Science field is completed. This demonstrates that the presented approach is suitable for bestowing the customisation of the Multilevel Scheduling allowed by pilot jobs to the final users, i.e. the efficient and easy profiting from grid and cloud infrastructures according to the computational requirements of their applications. The last item is also considered as an additional contribution that must be explained in appendices.

For these reasons, the structure of this thesis follows these previously mentioned points. Chapters 2 and 3 describe the state of the art, this is, the strengths and drawbacks of the existing scheduling approaches and frameworks, the reason for their design and the compiling of motivations to develop a new pilot system. Moreover, in Chapter 3, the specific division in subsequent chapters of the rest of the research is detailed and justified by the requirements of the new pilot system. Therefore, this paragraph only aims an overall vision of the structure of this work. Thus, the second point is achieved in the Part I of this thesis, which is focused on propose new approaches (Chapters 5 and 6) to be used for the early-binding scheduling of the calculations adapted (Chapter 4) to distributed environments. Finally, the architecture of the new pilot system as well as its design details that makes its features possible is introduced in the Part II. Therefore, the mechanisms to properly manage the different scheduling layers, the execution of legacy applications and the incorporation of third-party schedulers are explained and demonstrated through the Chapters 7, 8, 9, 10 and 11. Furthermore, in addition to the Chapter 12, in which a summary of the main intellectual contributions and future work is presented, this thesis concludes with complementary appendices that show the impact of the research performed on different fields (Appendix B) besides of the publications (Appendix A), in special new physical have been achieved for Nuclear Fusion (Appendix C).

Chapter 2

State of the Art

2.1. Distributed Computing paradigms

Computing systems, which elements maintain independent their memory from processors belonging to other elements, can be considered as distributed platforms. Storage can be considered as a type of memory. Consequently, following strictly the previous definition, a distributed system has its memory, processors, and disks physically distributed into isolated elements. As any other multi-processor system, to improve the execution of an algorithm, slices of calculation are distributed among different processors and the speedup obtained is strongly limited by the Amdahl's Law [33]. However, in contrast to the shared-memory platforms, the communication between two processes running on different elements has to be performed through the interconnection network. Therefore, it is the network latency, (and in lesser degree, the bandwidth) the key factor that limits the performance gain of parallel algorithms on distributed platforms. In this sense, the overhead introduced by communications usually increases with the number of processes and it can become unmanageable, being higher than the real processing time when conventional LANs (or WANs) are used. Additionally, the probability of failure increases with the duration of every computational slice. A failed process can stop and waste the whole parallel calculation or, in the better case, it will imply restarting this process, stretching on the final makespan. Thus, the reliability of every distributed element gains influence on the speedup with long processes, especially when these elements are uncontrollable. Other key factor is the level of heterogeneity of the elements, because the slower process can determine the final makespan of the whole calculation.

Precisely, the capacity for accomplishing parallel algorithms is the distinguishing feature of High Performance Computing (HPC) architectures from the High Throughput Computing (HTC) ones. HPC platforms tend to offer a uniform hardware access from all processors to memory, avoiding the effect of slower elements and communications. Therefore, HTC platforms are only suitable for accomplishing embarrassingly parallel algorithms or completely independent processes, where the order of execution does not influence the execution time of every job, only the ones on which dependences should be preserved. For this reason, their performance is usually measured in jobs per second (i.e. the throughput), while the HPC ones are in operations per second (i.e. their capacity for performing processor operations in parallel). It is noteworthy to mention that HPC platforms can accomplish HTC calculations, but the feature is

not achieved backward.

The HTC platforms are a sub-set of the wide range of architectures proposed for Distributed Computing and multiple aims. In this sense, diverse classifications can be found [34, 2, 35, 36, 37], but currently the accepted Distributed Computing paradigms for HTC can be summarised in Cluster, Volunteer, Grid, and Cloud Computing as well as Big Data, following the historical order of appearance.

Clusters are the evolution of the non-uniform memory access (NUMA) platforms for massive parallel processing (MPP) to commodity hardware. They usually support both HPC and HTC calculations. For this purpose, the job scheduling is managed by batch queues or local resource management systems (LRMS) inherited from the previous platforms. They implement shared storage area networks (SANs) as well as low latency networks such as Infiniband. In general, they are composed by identical servers based on x86 architecture supporting the same configuration (operating system (OS) and software releases). The cluster is within the administrative domain of an institution, where quotas and preferences are set up for the users according to the different projects of interest.

A step forward is the inclusion of the personal computers (PCs) of the institution campus in the resource pool of the cluster [7]. Resources still belong to the same administrative domain: they are connected to the intranet and supervised by authorised staff. However, as the PCs are not under the protection of the datacentre, they are exposed to power cuts, performance slowdowns or hangs, because the desktop is also running arbitrary programs and the intranet can experiment unexpected congestion. Moreover, every user requires a PC with certain configuration. Thus, the heterogeneity and the reliability make these implementations only suitable for HTC.

Unlike clusters, Volunteer Computing platforms distribute the calculations across different administrative domains, which are the PCs belonging to anonymous people around the world. Additionally to the privacy issues and the low network performance compared with an intranet, the reliability and heterogeneity are obviously much more important in these systems, because the resources escape the control from any administrator [38]. However, both Volunteer and Cluster Computing are centralised architectures since only a coordinator system is allowed to use their resources.

On the other hand, the Grid Philosophy proposed by Foster [1], defines a grid as a system (i) not subject to a centralised control, (ii) based on standard, open and general-purpose interfaces and protocols; that (iii) provides some level of quality of service (QoS). This definition is applicable to Cloud Computing and Big Data, but with the necessary extension to their peculiarities and features. In this sense, the Cloud Computing covers much more computational needs than the ones required for an HTC platform, like resource elasticity, service consolidation, or cost reduction [39]. These features are offered thanks to the virtualisation mechanisms of machines (infrastructure-as-a-service, IaaS clouds), development tools (platform-as-a-service, PaaS clouds), and applications (software-as-a-service, SaaS clouds). The term Big Data generally encompasses the inherent difficulties of storing and mining the great amount of data generated by scientific or industrial experiments, business accounting, social networks and even, the whole internet activities. However, the meaning of interest for this work is the one related to the mechanism to split and process pieces of the data into multiple providers, which offer specific interfaces for data-mining algorithms such as MapReduce [40]. Therefore, Grid, Cloud and Big Data paradigms propose a service oriented architecture (SOA) [41], where any user can benefit from computational resources with certain QoS without the needed passing-

through a unique and centralised system.

This work is focused on the efficient scheduling of HTC calculations in large grid and IaaS cloud infrastructures. The abstractions enabled with PaaS and SaaS clouds are counterproductive for HTC scheduling because they do not allow the characterisation of resources at infrastructure level. However, these and other paradigms are closely related and usually are mixed through the related work. For example, grid resources are customary formed by clusters and can elastically grow with clouds [42, 43]; or the algorithms used in Big Data can be deployed on clouds [44, 45] and even offered as-a-service (PaaS and SaaS). Therefore, the interest is not to explore the approach usually called as Utility Computing, where one (or few) provider on-demand allows increasing the computational power of certain institution. The aim is going further by studying the effective distribution of the calculations among the resources from a large set of grid and IaaS cloud providers.

For this purpose, the necessary concepts to understand how the calculation workload can be scheduled in grid and IaaS cloud infrastructures following the SOA model are summarised in this chapter.

2.2. The grid and IaaS cloud approaches

The main achievement of the research in Grid Computing was the establishment of a service oriented architecture (SOA) [46] for HTC, which was widely accepted by the whole community. This fact allowed the federation of large volume of resources across the world, but also implied a long process of testing and standardisation. One of the results was the creation of a set of APIs for highly distributed computation [47], as well as the establishment of protocols to interface with the computational services offered. However, Grid Computing has not properly addressed several problems. One of them is the rigidity of configurations that are present in the federations. Besides, one of the major issues in grid computation is still the efficiency of the submitted jobs. Such efficiency can be considered from different perspectives [6], but always bearing in mind that the final users want their calculations ended in the shortest possible time. For this purpose, it is mandatory to count on scheduling mechanisms that properly build and distribute these jobs among available providers.

Cloud Computing promises to be more simple, flexible, usable and available than Grid Computing. Nevertheless, the latter affirmation is far away from being a reality in many cases. In first place, the diverse sponsor institutions, funded projects, infrastructure providers and manufactures have different views, and propose different models about how the cloud federation should be. This is, Cloud Computing is experimenting a process of standardisation and consolidation similar to the one performed for Grid Computing. A good introduction of this matter can be found in [48]. Due to this diversity, the result is an increased complexity of the current cloud platforms from the user's point of view. On the other hand, although the flexibility is increased, users not always can run their applications exactly on the virtual environment that they require, and resources are effectively limited for every user. Additionally, they usually lack of APIs and service interfaces similar to the ones available in grids. As a result, optimising the placement of virtual machines (VMs) across multiple clouds and also abstracting the deployment and management of components of the virtual infrastructure created are complex. Such plethora of solutions were surveyed and classified in the proposed taxonomy by Grozev and Buyya [49].

Regardless, any scheduling approach working on any optimisation [6] aspect should take account of localisation and performance of the available resources to target specific providers. Obviously, computational resources must be requested by interfacing with the services that providers offer, but these resources must be previously discovered and characterised in some way. Therefore, an adequate introduction to the design and behaviour of current grid and cloud infrastructures should begin with the description of these protocols and services.

2.2.1. Interfacing with grid and cloud providers

2.2.1.1. Grid interfaces

These interfaces have been traditionally tied to the different middleware implementations. With the development of Globus Toolkit [50] the establishment of first grid infrastructures was possible. Consequently, GRAM2 [51], was widely accepted as the gatekeeper interface for computational jobs, GASS and GridFTP [52] protocols were used for transferring their inputs and outputs, the security was based on X.509 certificates and the description of sites on LDAP[4], among other services.

Although other middleware such as EDG/LCG/gLite/UMD/EMI [53], ARC [54], GridSAM or UNICORE [55] progressively offered another interfaces for other purposes, the basic operation with grid sites relied on the protocols stated by Globus. However, this *status quo* ended with the introduction of web services (WS) into the architecture of Globus 3 and 4 [46], but the presented protocols were also not standardised. Consolidated infrastructures are resistant to change to the new middleware, maintaining old interfaces until the lack of support forced their developers to implement new ones. Finally, Globus 4 resulted unfeasible, but web services were adopted. Currently, grid gatekeepers are mainly based on CREAM [56] or on the standardised OGSA-BES [57] protocol, while storage is usually accessed through GridFTP or SRM.

2.2.1.2. IaaS interfaces and contextualisation

The need of specific interfaces that abstract the common operations with VMs (creation, booting, stopping, halting, destruction, etc.) has driven the appearance of several proposals and implementations (such as Globus VWS, Nimbus, Eucalyptus [58]) since 2006. However, Amazon Web Services[1] (AWS) was the first large IaaS provider and many deployments were based on its interfaces, because it was considered as the *de-facto* standard for the industry. Lately, the Open Grid Forum (OGF) standardisation group proposed the OCCI [59] and, in general, is supported by a wide set of current virtual infrastructure managers (VIMs) such as OpenNebula[2], OpenStack[3], or Synnefo[4].

Other issue is the need of instantiate VMs with a certain configuration. The initial approach is to upload the customised disk images of the VM to the provider. Nevertheless transfers are too expensive due to the size of images. Therefore, it is more efficient the support of generic VM templates at every provider. Thus, the *contextualisation* is the procedure to pre-configure a VM at boot time. In this sense several technologies

[1] http://aws.amazon.com
[2] http://opennebula.org
[3] http://www.openstack.org
[4] http://www.synnefo.org

have been developed, many of those tightly dependent on a concrete VIM. Finally, Cloud-Init[5] is imposing around the current IaaS providers.

Nevertheless, the interfaces and contextualisation tools are not enough to completely manage a virtual environment. Other services and systems are needed, especially when multiple cloud providers are available.

2.2.2. Multiple providers versus federations

There is a conceptual differentiation between a simple group of providers and the ones making a federation that has important implications on the feasible scheduling to be performed. Following several definitions in the related work [49, 60], when a client (or service) uses multiple, but independent, and not related grid or cloud providers, he is working on a multi-grid or multi-cloud environment. Therefore, it is the client (or service) who must completely manage the compatibility among interfaces, monitor every provider, and handle its authorised accounts because it is working on different configuration and security domains. This entails a lot developing work that should limit the scheduling capacity of the client system. In consequence, the distribution of the calculation among providers will scale on the order of few orders of magnitude, although these providers can supply a great amount of resources (e.g. AWS).

These difficulties are widely studied in the related work, and they are usually presented as an interoperation [47] issue among grid or cloud infrastructures. For example, the interoperation of grid islands [61] (or multi-grid [60] environments) has been managed through the tools described in Subsections 2.4.2 and 2.4.3. Additionally, the multi-cloud [49] approach currently is too common due to the multiplicity of cloud conceptions and commercial providers. Interoperation among clouds can be faced with similar tools as grid but entails the same drawbacks.

To enable scheduling systems for managing providers on the order of thousands, these systems should work on federated infrastructures. Federations voluntarily associate providers, which even share their resources among each other, but completely following the SOA model [41]. In this sense, the weaknesses of multi-grid and multi-cloud approaches are not related to deal with the interoperability issues that SOA tackles. Thus, it is not enough to offer services as the ones described in previous Subsection 2.2.1. These services must accomplish common visibility, governance, security, orchestration and monitoring properties among others. Therefore, it does not simply imply the agreement to use certain protocols; it also includes the establishment of common services as:

- Information systems (IS): they are indexation locations where the rest of services are dynamically described. They constitute the starting point from which any client (i.e. the scheduler) can discover the resources belonging to the federation. The development efforts to characterise interesting aspects of the infrastructure should be focused on this service.
- Authentication and authorisation infrastructure (AAI): it stand for the group of services, authorities and procedures that enable the security governance in federated environments, which are usually based on encryption and temporal tokens. They work together as an overlapped and independent infrastructure that allows the management of users and projects within, or even crossing federations, i.e. it allows the establishment of virtual organisations (VOs). Through

[5] http://cloudinit.readthedocs.org

AAI, the clients or groups of clients are granted to use certain amount of resources by setting quotas for them in providers according to signed contracts.

- Accounting, monitoring and incident systems: they compile the performance, throughput and failures of every provider and every user through the time. Therefore, they offer a detailed measurement of the current QoS of whole infrastructure that can be useful for scheduling, not only to check the compliance with SLA contracts subscripted by providers. In this sense, the information should be also summarised in IS for further benefit. Additionally they include the procedures of notification and solving from the issues detected.

2.2.3. Federations established

The paradigm of a federated infrastructure is the grid constructed to process the data generated by the Large Hadron Collider (LHC) experiments at CERN. It was built throughout: the consecutive core projects European DataGrid (EDG), Enabling Grids for E-SciencE (EGEE) and currently the European Grid Infrastructure[6] (EGI); the satellite ones EELA-2 and GISELA, among others, which were progressively assimilated in the main infrastructure; the association with Open Science Grid[7] (OSG), which is the proposal of the United States of America to create a large federation; and, other infrastructures such as NorduGrid in Scandinavia. The initial infrastructure devoted to High Energy Physics (HEP) was progressively turned into a multi-propose platform counting on more than 530,000 processors and 500 PBs of storage, opened to any scientific area.

Note that EGI, OSG or NorduGrid are considered as different infrastructures, i.e. they has their own agreements about the middleware to be used, the QoS provided, etc. Thus the global platform is actually a federation of infrastructures, each of which is a federation of providers.

The goal was achieved by deploying common services with the properties described in the last subsection:

- The top Berkeley Database Information Indexes (BDIIs) as IS. They are LDAP servers that compile the characterisation of providers structured following the GLUE schema. Several can co-exist in an infrastructure. They can also be hierarchically organised, filtering the information of BDIIs bounding sub-infrastructures. It is noteworthy to mention that other mechanisms were tested, for example the WSRF of Globus 4 in OSG, but finally were discarded.

- The establishment of an AAI based on signed X.509 certificates, distributed certification authorities (CAs), temporal and delegated proxies, and VOMS as authorisation service.

- The deployment of monitoring systems to regionally and globally test the infrastructure, which trigger several notification and tracking actions. Currently, SAM tests are managed with Nagios, and incidents are tackled with the GGUS ticket system. Accounting is performed through APEL.

On the other hand, some associations of providers cannot be considered federations according to the definition proposed in this work, in particular, the partnership

[6] http://www.egi.eu/
[7] http://www.opensciencegrid.org/

of public HPC centres. While infrastructures such as GARUDA[8] follow similar federation procedures than the ones described for EGI or OSG; XSEDE[9] (as continuation of TeraGrid) and PRACE[10] (as continuation of DEISA) do not deploy IS or VO services, they do not even offer interfaces such as the described in Subsection 2.2.1. The execution access is managed through centralised user interfaces (UIs) which maintain the authorisation credentials of users. In general the UNICORE graphical interface is used, or the UI directly enables the possibility of opening remote SSH sessions to directly post LRMS commands in the HPC facilities. Additionally, a mix of clustered file systems such as GPFS and Lustre are used as global storage. Thus, the utilisation of GRAM5, OGSA-BES or GridFTP is residual.

With respect to cloud infrastructures, several initiatives have been proposed through the last years [62, 63], but the long-stablished one is EGI FedCloud. This federation is taking advantage of grid experience to deploy grid-style services to enable its federation[11] [64]. Cloud sites fully support the EGI AAI based on X.509 and VOMS, sharing the same VOs already established for grid. Providers must be compatible, at least, with OCCI, but also expose their characteristics by LDAP to be compiled by top BDIIs. This last aspect is very important to establish a real federation. OCCI shows information about VM templates, allowed resources, etc. However, to perform the discovering of new cloud sites or to facilitate their monitoring, any system should have access to an IS. In the same way, the accounting, monitoring and incident tracking was managed with the same tools than EGI, but adapted to the cloud environment.

2.3. Scheduling in large infrastructures

2.3.1. Definitions

The *Workload Scheduling* is the process for which the calculation is split into manageable units and distributed among a set of *known* resources. However, there are two main approaches to obtain these resources in large federations: the *early* and *late-binding* methods.

Like to the code compilation process that fixes the assignation of the object type to a variable (or function) before running the application, the *early-binding* scheduling fixes the slice of a calculation to a unique provider before this provider has supplied the corresponding resources. This is, the selection of the provider is performed before the job was submitted.

In contrast, the type of variables is known at runtime following the *late-binding* approach (e.g. scripting languages). In the Distributed Computing field, late-binding denotes that the resource is supplied before any assignation of workload was performed. In consequence, a new level of scheduling is added, which is not directly managed with Workload Scheduling: the *Resource Provisioning* level. This implies the implementation of resource appropriation mechanisms such as the pilot job technique, but provides the biggest possibilities in terms of low overheads and flexibility [65].

Workload Scheduling and Resource Provisioning are the basis of the *Multilevel Scheduling* considered in this work. Obviously, both can be performed with diverse software, which are stacked to several scheduling layers. In this sense, part of the

[8] http://www.garudaindia.in/
[9] http;//www.xsede.org
[10] http;//www.prace-ri.eu
[11] https://www.egi.eu/infrastructure/cloud/

Workload Scheduling, such as the division of the calculation, is commonly performed by user's tools and it can be called *User-level* or *Application Scheduling*. However, to completely profit from the Multilevel model, every layer should communicate and influence the others.

On the other hand, leaving some collaborative [66] scheduling aside, the early-binding approach does not limit by itself the deployment of a wide range of scheduling algorithms on grids and clouds. The actual reason is that middleware lacks of the properly characterisation of resources according to the application requirements. To limit its negative impact, several strategies have been followed, such as the dynamic allocation of jobs depending on the infrastructure status and capacity at any time [67].

To understand the possibilities and issues that really offer every approach, the type of HTC applications suitable for being distributed among grid and cloud providers must firstly be clarified. This includes how their workload can be split, how the providers are selected, the resources are used, and finally, which tools are available to perform them. The following subsections offer an abstracted view of these aspects, while current frameworks are deeply explained through last sections of this chapter.

2.3.2. HTC workloads feasible to be distributed

First the latency of WANs, and then, the data-locality, the reliability and the heterogeneity of resources, mainly limits the suitability of algorithms to be distributed among grid and cloud providers. These affirmations are fully justified in Section 2.1. In this sense, HTC codes can potentially run on grid and cloud by definition, but some of them are more adaptable than others, even if a portion is unadaptable in practice, especially on grid environments. For example, it is unfeasible to execute codes which non parallelisable sequential part lasts more than 48 hours, since grid sites do not usually allow this walltime in their queues. Other example is those applications that continuously require reading access to a large amount of indivisible data, being the computation time insignificant with respect to the transferring time.

The feasible calculations are the ones which main workload (calculation and data) can be completely or partially divided, being more adaptable when more divisible are, because the obtained parts can be arbitrary distributed. This feature is usually denoted as malleability or *mouldability* [68]. Obviously, the calculation can contain sections that should be executed in an HPC environment. These elements increase the complexity of the adaptation and it usually decreases the performance obtained, because some providers have to support them (which is not common). Moreover, other issues such as the imposed middleware or the libraries required can hinder their adaptability. All these aspects are summarised for grid in Figure 2.1. In the case of cloud infrastructures, most of configuration issues are avoided thanks to virtualisation.

Therefore, the more suitable calculations are: bunch of independent computational sections (random numbers, parameter sweep, or mixes of both) and computational workflows (composed by several instances of the first ones).

The main examples of algorithms based on random numbers are the Monte Carlo (MC) methods [69, 70]. They are a class of algorithms employed for modelling complex phenomena which are often used in physical and mathematical problems due to they are well suited for problems where it is very hard to obtain a closed-form expression or unfeasible to apply a deterministic algorithm. The simplest type of MC algorithms allows the division into computational sections that correspond to one random generation (one simulation or sample) and subsequently, they are suitable for running on

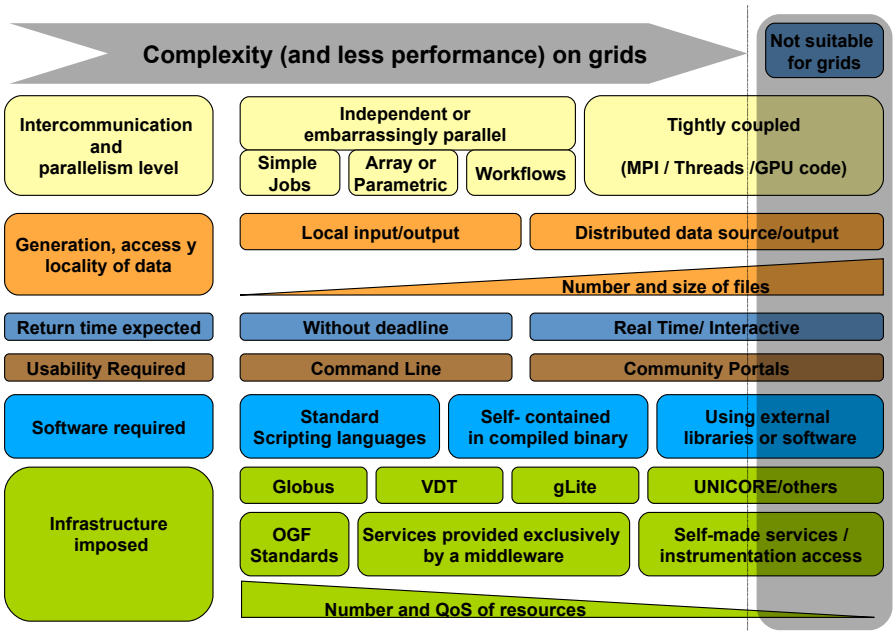


Figure 2.1: Suitability of applications for being executed in grid environments.

grids or clouds. In this sense, this type of MC is applied to problems of very different nature, such as Radiological Medicine [71], Chemistry [72], Plasma Physics [73], or HEP [74], and they require to be solved in a manageable interval of time despite their high computational load. On the other hand, parameter sweep calculations are the ones based on varying their parameter values within certain ranges, and they are also typical on grid [75].

Consequently, every simulation randomly generated is named as *sample* and every combination of parameters is called as *task*. The mouldability property allows grouping several samples into a single task because they share the same input files and combination of parameters. In the same way, mouldability allows grouping several tasks into a bag of tasks (BoT) that usually share the same inputs (although it is not a necessary condition).

The distributable computational unit for early-binding scheduling is the *job*. It is composed by a unique BoT, but it also specifies a set of requirements for its correct execution. These requirements include preferences or constraints related to any characterisation aspect such as hardware, software, protocols, data locations and even degrees of reliability or QoS.

Workflows are a set of jobs that must preserve an order of execution and termination because the outputs of some jobs are the inputs of others. Usually, the most usual and manageable type of workflows is the direct acyclic graph (DAG) [76], but there are other models such as Petri nets [77, 78], or other abstractions that support the provenance [79] and reproducibility of the data processing. In any case, the execution sequence can be also set as precedence constraints within the requirements of every job. Additionally, the data-allocation constraints are usually related to workflows because can be used for provenance, but these requirements are not restricted to them [80]. Jobs without precedence constraints are widely used to compute large

sets of data stored, the localisation of which becomes a scheduling issue [81].

2.3.3. Workload Scheduling

Grids and clouds are complex systems [82] due to their dynamism and heterogeneity and, consequently, calculating the best match between a set of computational requirements and resources in an effectively characterised infrastructure is, by definition, an NP-complete problem [3]. Thus only sub-optimal schedules can be found. This affirmation can be extended to clouds because although the virtualisation mechanisms allow instantiating customised execution environments, every provider in a federation effectively supplies a variable amount of resources, which are supported by different hardware and achieve different performance and QoS. Moreover, cloud federations will be more dynamic than grids because much more parameters are taken into account for scheduling [83, 84].

Having in mind these facts, multiple scheduling approaches can be followed. In general, several approaches developed in Operative Research area [85] used to improve the industrial manufacturing, transportation or telecommunications can be applied. For example, the Queue Theory [86, 87] is a reference point to take account of a wide range of characteristics and requirements. Graph Theory [88], and in special, the Theory on Directed Graphs [89] can be useful to improve the execution of single and concurrent workflows. Moreover, classical stochastic [90] and probabilistic [91] methods, used to estimate the flow of requests, can be mechanisms to foreseen busy providers and reduce the global cost in market-oriented environments.

However, it is noteworthy to introduce the specific approaches proposed for the HTC applications described in previous Subsection 2.3.2. In this sense, taking account of the mouldability property of workload, an excellent mathematical approach is the Divisible Load Theory [92] (DLT). A good introduction is the work of Robertazzi et al. [93, 94], which methodology is also applied to data-loads [95] or to cloud resources [96]. Equivalent MC methods can be used [97] rather than DLT. Furthermore, multiple self-scheduling approaches also take advantage of mouldability. For example works on loop-based algorithms [20], or about the adapted procedures from the Artificial Intelligence area, such as genetic algorithms [22] or heuristics [21], can be cited as useful comparative studies.

On the other hand, a job must be constructed for every BoT. Besides the specification of the location and names of outputs to be distinguishable among the results from other executions, the building process includes the establishment of a requirement set for the job. This latter action must be considered another type of scheduling because allows several ways of optimisation. In this sense, the requirement set does not only contain the preferences among types of resources, it includes the precedence constraints among jobs that make up a workflow, which can be improved with diverse techniques [98, 99, 100]. Job specification also includes the URIs of several suitable inputs and the adequate place to store outputs, which also implies further researches [80, 81].

Subsequently, the final scheduling phase is to correctly distribute jobs among providers [101], taking into account their requirements. The scheduling system can be in charge of distributing many jobs belonging to different applications and users. Therefore, the advanced algorithms implemented in commercial LRMS such as backfilling [102] can be useful for grid and cloud. However, in contrast to cluster environments, the job scheduling includes the previous discovery of providers and their

characterisation, which is indispensable to their correct selection.

All of those scheduling approaches and their consequent combinations are difficult to be classified. In this sense, several taxonomies can be found through the literature; thus, the typical generic ones such as [103, 21, 104], or the more specifically focused on workflows on grid [105, 106] or on cloud [107], are extensive introductions. However, they do not provide with a proper focus on the Workload Scheduling that allows illustrating the behaviour of the available tools and the work achieved in this thesis. To solve this issue, a simplified vision of the Workload Scheduling is proposed. It implies its differentiation in three scheduling fields:

Workload Division. It stands for the procedures related to creation of BoTs. In this phase the size of every BoT is chosen before any execution. In contrast to other authors [20, 108, 109], this scheduling is considered *static* when these sizes are pre-established by a mathematical function without taking into account the infrastructure status. Thus, they include the equal-size divisions as well the decreasing-size ones. On the other hand, when the scheduling algorithm changes the size during the calculation according to the infrastructure status or the statistics retrieved from current executions, it is considered as *dynamic*.

Job Building. For every BoT a distributable job is completely described and formatted with certain specification or language (see Subsection 2.3.6). The scheduling field is related to the establishment of requirements, constraints and preferences that enable a previous optimisation and a posterior guidance of job execution. The process is considered *static* or *dynamic* following the same meaning exposed in the previous phase, i.e. if it takes account of infrastructure status.

Job Scheduling. This scheduling field stands for the job scheduling and the effective submission steps explained in [101], namely: resource discovery and selection according to the requirements set in the previous phase, job preparation, submission, monitoring, migration and termination.

Note that the three fields include a phase of characterisation of the infrastructure when *dynamic* scheduling is performed. Moreover, the characterisation phase must be completed before the matchmaking process in Job Scheduling. This issue is repeatedly commented through the text, but especially in Subsection 2.4.2.2, where certain tools (*suppliers*) are separated from the ones that properly perform cloud brokering. In this sense, resource brokers (RBs) are the main entities which handle jobs but a similar scheduling approach is performed by systems managing VMs in cloud federations. Additionally, the selection of a certain provider can be performed by other tools working on the Job Building field, due to these systems can constraint the job execution on this provider. For example, when workflows are directed by workflow managers (WfMs), the RBs will usually only perform the submission and control of the job, i.e. working as *dispatchers*. Workload Division is usually performed by application-level approaches such as the self-schedulers. However, some self-scheduling frameworks deal with the whole Workload Scheduling. These three main types of tools (RBs, WfMs, self-schedulers), their scheduling domains and combinations were deeply explained through the Section 2.4.

2.3.4. Turnaround model and characterisation for scheduling

Before describing more deeply the tools and approaches available to perform the Workload Scheduling, the basic model that supports any of the scheduling algorithms

compiled in this thesis must be introduced, i.e. the job turnaround. In this sense, the widely accepted and simplified definition of turnaround [68, 110] follows the equation:

$$T = T_{sched} + T_{xfer} + T_{exec} \tag{2.1}$$

where:

- T_{exec} is the effective execution time of the application in the remote resource. It depends on the hardware, software, overload and reliability offered by the provider for the concrete application, as well as on the BoT size.
- T_{xfer} stands for the time wasted in transferring inputs, outputs and software. It depends on the protocols used and the bandwidth of the remote provider, as well as on the amount of data to transfer.
- T_{sched} comprises the time wasted in Job Scheduling processes. It depends on the infrastructure overload (queues and shared RBs), the allowed utilisation (quotas) and the middleware implementation. The Workload Division and Job building are only included if they are performed by the same system, i.e. a WfM or self-scheduler; if not, the added overhead is usually denoted as related to the application behaviour (T_{app}).

Consequently, the suitability and preference for any provider depends on its *real availability* for a type of calculation at certain time, which is compound by:

(a) the volume of the hardware and software resources actually offered, i.e. number of cores, memory, storage, bandwidth, data, libraries, licenses, etc., as well as the VM images in cloud providers;

(b) the effective performance of these resources for the application, i.e. the profile of the application on that hardware, as well as the granted percentage utilisation (because the hardware is usually shared) and the potential associated costs (especially with cloud providers).

(c) the reliability of the application, the middleware and the provider itself (that depends on the configuration, maintenance, security, network supplier, etc.).

Nevertheless, as the date of writing this thesis, the information belonging to point (a) is as much partially available at the IS deployed in federations. The rest of points are practically discarded. In particular, the GLUE schema implemented in top-BDIIs currently does not allow knowing (among others aspects):

1. the estimated waiting time for a queue at the LRMS of a grid site;
2. the capacity and filling rate of a cloud provider;
3. the maximum number of jobs that certain user can queue and run in a grid site;
4. the quotas established in cloud providers for a user belonging to certain VO;
5. the exact hardware or reliable benchmarks of every worker node (WN), group of identical WNs or virtualisation hosts;
6. statistics about the QoS or the SLA achievement for every provider.

It is noteworthy to mention that GLUE is extensible, and currently count on tags that declare some of the aforementioned characteristics. Nevertheless, the information provided is usually incomplete, not updated and erroneous in many cases. Additionally, much more aspects should be taken into account in cloud environments [83]. Nevertheless, the grid experience with GLUE shows that many characterisation aspects will never be implemented. Obviously, a scheduling system working with cloud providers can follow a multi-cloud approach, monitoring every one through its OCCI interface. However, the OCCI v1.1 specification does not even provide the aforementioned information.

Moreover, accounting and monitoring tools do not aggregate their statistics to the IS, and also cannot be directly interfaced because only visual tools are available [111]. In this sense, standardisation advances are carrying on [112], but there are not implementations to evaluate their suitability.

Therefore, the deployment of advanced Workload Scheduling algorithms are hindered by this lack of characterisation required to estimate the job turnaround. Having in mind the aforementioned issues, the possibilities that systems performing early-binding really offer will be described through the Section 2.4. In contrast, late-binding techniques can solve many of the commented characterisation problems and they can provide more advantages, as will be explained in the following subsection.

2.3.5. Resource Provisioning with pilot jobs

The current necessity of a grid job to access different grid services anywhere outside the specific administrative domain of resources where it is being run implies that this job could require access to the Internet through NAT or proxy services. This requirement enables any master-slave application where the slave, running inside a worker node (WN), to directly communicate with a coordinator server through its own communication protocol and bypass (some) grid middleware. The main motivations for establishing this network overlay within a DCI are:

- to reduce overhead, mainly the waiting time in remote queues at the LRMS and shared GSs using slot appropriation;
- to effectively characterise the assigned resources (the WNs) by sending their real properties and current status to the master;
- to enable compatibility with other legacy systems and applications by checking and creating special configurations; and
- to reduce grid complexity by directly using assigned resources and monitoring the user tasks.

Thus a pilot system [17, 19, 23, 24, 25, 26, 27, 28] is defined as a framework where many satellite programs (pilots) running inside grid jobs branch out to monitor a set of computational user tasks (or BoTs) that are continuously assigned by a server following the master-slave scheme. The difference between user task and grid job must be introduced; in this study, a pilot job is a regular grid job, and each of the computational parts of the user application that are being run inside a pilot job are user tasks. In this sense, the series of procedures needed to set every pilot job up is within the scope of Resource Provisioning, as well as the mechanism to assign every task to pilots is within the Workload Scheduling. Subsequently, two of the scheduling fields that belong to the later are renamed as Task Building and Task Scheduling.

On the other hand, the provisioning mechanisms currently available in IaaS clouds are focused on setting up VMs for the consolidation of services. However, they do not prevent running pilot jobs as temporal services [29, 30, 31]. In general, for the most of those systems that demonstrated their suitability in grid [113, 114, 115, 31, 65], the use of cloud resources will provide the following advantages:

- Concurrent use of clouds and grids (and other infrastructures such as local clusters or Volunteer Computing if supported by the system).
- Compatibility with applications previously ported to these systems, at least, preserving the achieved performance or speedup.
- Keeping the expertise acquired from grid, reducing the user training-gap and the operational costs.
- Preserving the robustness of the system or the use of complementary tools such as web browsers, monitoring tools, etc.

However, the utilisation of current pilot systems also implies drawbacks. The weaknesses are the consequence of their design, mainly if the system adopts a pushing or pulling behaviour. The former implies technical issues that hinder their deployment, while the latter strongly condition a feasible scheduling [65]. Moreover some of them are designed for a concrete calculation type and they lack compatibility or adaptability. Furthermore, the performance of most of them is unpredictable, which prevents coupling them with Workload Scheduling algorithms. For this reason, few abstraction models of pilot jobs have been proposed [12, 116, 117]. All these issues are deeply explained through the Section 2.5.

2.3.6. The developer's point of view

Other important issue to be tackled is the mechanisms to code the applications to be distributable. In general, developers implementing embarrassingly parallel calculations should distinguish between two complementary concerns [118]: the *computation*, i.e. the procedural mechanism to execute a job; and the *coordination* of the jobs generated. Thus, the submission and monitoring of jobs are included in the *computation* issues. Unlike, the Workload Division, the Job Building and the selection of resources performed within the Job Scheduling field belong to the *coordination* problem. This is, *coordination* corresponds to the User-level (or Application) Scheduling mentioned in Subsection 2.3.1. Moreover, any entity (application, self-scheduler, WfM or RB) that interfaces with other system makes this differentiation. This is the reason for isolating the submission phase in the simplified structure of self-schedulers described in Subsection 2.4.4.

In particular, developers count on several programming models and tools which progressively avoid the need of writing too much code for the *computation* issues, and more advanced ones that facilitate their work on the *coordination*, i.e. the real scheduling. Therefore according to the abstraction level provided, any developer must choice among the approaches described below:

(a) To directly **implement the interface protocol** of the target system. The application developer codes the binding with the services of resource providers such as GRAM, OGSA-BES or CREAM, with the IS of the infrastructure, or with

other systems, such as a WfM or a RB, to delegate some scheduling procedures. This approach is expensive and prone to errors. It should be only selected in the absence of other solutions.

(b) To perform system calls on a command line interface (**CLI**) to perform local or remote operations on providers or other systems. It is lesser expensive that the pervious approach, but inefficient, increasing dramatically the overheads, and tied to middleware.

(c) To **make use of an API for *computation***. Applications are much more robust using procedures from a library than system calls to commands. There are two options:

a) **Middleware-specific libraries**. Globus, gLite or GridSAM libraries are available for interfacing with services such as the aforementioned ones, but they tight the code to middleware.

b) **Widely accepted APIs**. The standardised APIs for HTC such as DRMAA [119] and SAGA [120] are the recommended way to deal with the job distribution. RBs and LRMS usually support DRMAA libraries and consequently, users will not need multiple versions of their applications for every system. However, DRMAA is focused on delegating the Job Scheduling to these systems and it does not allow querying the IS, neither managing data, etc. Unlike, SAGA is continuously extended with adaptors which progressively enable the compatibility with any service, such as the ones used in cloud. Additionally, other libraries that expose a non-standardised API should be also considered. For example, Ganga [121] also provides similar portability among platforms and it is widely used in HEP calculations.

(d) To **abstract the *coordination***. Previous approaches progressively hide the access to resources, but the developer must deal with the Workload Division and the Job Building without assistance. Consequently, specific languages, patterns, templates or translators have been proposed to soften the *coordination*. In this sense, a large compilation of the approaches proposed for parallel calculations can be found in [118]. Although some of them are feasible for distributed environments, it is necessary to extend the classification with the proposals devoted to HTC.

a) **Workflow specification languages**: They allow specifying a complete workflow that can be globally optimised by an underlying system. In general, RBs only support simple DAGs, and some WfMs works with their own description languages. However, more complete and general-purpose specifications for workflows were proposed such as WS-BPEL [122], GSFL [123], AGWL [124], SWFL [125], GWEL [126], or the YAWL [127]. Moreover, some are based on coloured Petri nets such as the GWorkflowDL [128] and the homonym GFDL [129]; or even with the UML [130] representation.

b) **Skeleton approaches**. They are high-level programming mechanisms which enable methodologies to abstract the scheduling algorithmic structure. They allow programmers to focus on some aspect of scheduling by tuning the workload algorithm offered by the skeleton framework or library. They include algorithm templates, design patters, procedures in imperative [131] or functional languages, or parameterisable scheduling tools. For example,

self-scheduling frameworks [132, 67] which support several applications are within this last category.

c) **Embedded directives**: In this case, the objective is not to modify the original code. Loops with high computational load or data processing are previously marked with an added instruction to be split into jobs and subsequently distributed [29], in a way similar to the OpenMP directives used for the automatic parallelisation of loops on shared-memory environments.

2.4. Instruments to perform early-binding scheduling

2.4.1. Scheduling domains and tools

Systems with scheduling attributions continuously generate and modulate the workload into different types of objects in grid or cloud environments. These systems only work with a volume and type of objects and relations belonging to one or several of the aforementioned fields, i.e. they works on a concrete scheduling domain to achieve certain optimisation objective. In this sense, systems in charge of Workload Division generate *equal* or *variable* size BoTs every interval time, independently if these BoTs are dynamically modelled or not. In general, the aim is to reduce the makespan of a single application, which is the scope of self-schedulers embedded in applications, as well as the self-scheduling frameworks.

Workflow managers (WfMs) are developed to facilitate users the management of data and the implementation of precedence constraints among their jobs. Therefore they mainly work on the Job building field, but they can also merge jobs to improve the graph. Both self-schedulers and WfMs typically can also modify the job requirements to fit the execution to the status of the infrastructure. Therefore they control the suitability of providers, and they can allow improving costs and energy, and achieve deadlines more properly than the simple Workload Division. However, they generally work with a single calculation with exception to the workflow engines (WfEs), which allow the concurrent management of workflows belonging to several users. Thus, fair-share policies and global throughput are partially implemented in these systems, as will be explained in Subsection 2.4.3.

Furthermore, a scheduling domain including jobs with multiple requirements is similar to a multi-user environment. Resource brokers (RBs) are designed to manage a large volume of jobs in these environments to improve the global throughput, resource usage and to grant deadlines and fair-share. In contrast to WfMs and self-schedulers, they only work on the Job Scheduling field because they must not modify the requirements that users set for their jobs. However, private RBs can potentially be deployed for a specific calculation, and consequently they can be tuned to reduce makespan.

Figure 2.2 shows relation of the scheduling fields in Workload Scheduling through the type of objects managed by the usual tools used in in early binding scheduling. As can be seen, self-scheduling frameworks manage all type of objects, but generally belonging to a single calculation. On the other hand, the scheduling domain of WfMs does not include jobs with fixed requirements, because a workflow implies that some jobs have constrained to the completion of other jobs. Job Scheduling is in the scope of self-scheduling frameworks and WfMs because many of them previously performs the

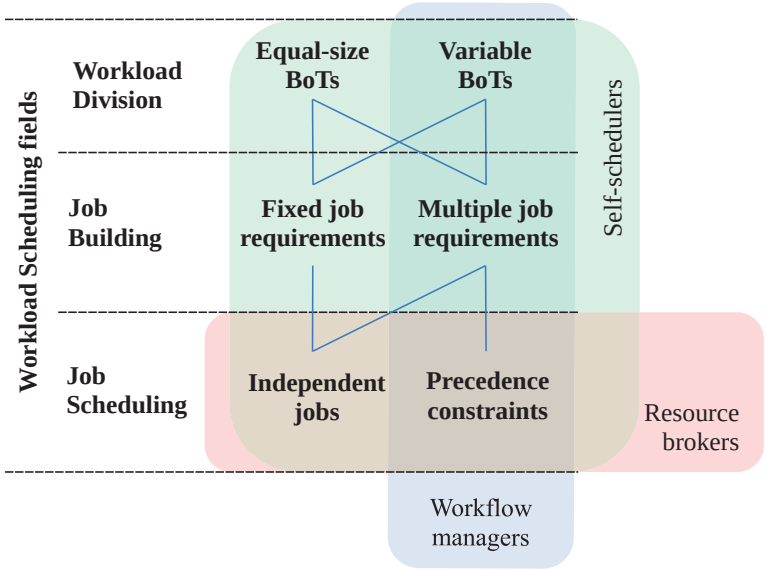


Figure 2.2: Workload Scheduling fields and the usual scope of scheduling tools.

matchmaking among jobs and providers and then directly interface with these selected providers. Obviously this is an expensive approach that limits their compatibility; for this reason, they rely on libraries for inter-operation (see Subsection 2.3.6) or even RBs are used as dispatchers.

2.4.2. Resource brokering in grids and clouds

Resource brokers automatically perform the matchmaking among jobs and resources following the typical Job Scheduling steps described in Subsection 2.3.3. Therefore, they are cognisant of grid and cloud dynamism, and they constitute the first mechanism to reduce the complexity of infrastructures, thus providing an automated and unattended access to all the resources. In this sense, the *resource broker* designation describes more properly the work performed by these systems. Through the related work, other terms have been used, but they suggest confusing meanings. For example, these systems have been typically called *distributed resource management* (DRM) systems, *workload management systems* (WMSs), *meta-schedulers* or *super-schedulers*. RBs do not necessarily have to manage the resources as if they were of their own, neither to mould the workload, nor to work on several infrastructures at the same time, nor to be stacked over other scheduling systems. These can be valuable features, but they get the concept away from the generality.

In this work, two RBs types are distinguished. *Grid schedulers* (GSs), specifically designed for the distribution of jobs in grid environments, and the IaaS *cloud brokers*, devoted to the placement of VMs across different providers. Obviously, many cloud frameworks completely manage the VMs instantiated as own resources; these provisioning mechanisms are depicted in Section 2.5, but the interest now is to introduce the systems that can be used for selecting the suitable provider before the VM is booted, i.e. the scheduling following the early-binding approach.

2.4.2.1. Grid schedulers

The current GSs only include fair-share mechanisms based on priority *first come first served* (FCFS) queues that enable dead-lines. Thus, the scheduling is focused on the global throughout and only DAGs are allowed. The reason for which GSs do not include more advanced algorithms is that their scheduling is based on the data published by the IS. As it is mentioned in Subsection 2.3.4, this information is not trustworthy and prevents to properly characterise the resources. However, this behaviour can be improved with diverse techniques. For example GridWay compiles statistical data about the successful executions, the waiting, execution and transferring times, which can be used for the subsequent selection of providers. Nevertheless, the absence of mechanisms to effectively profile the applications and to reserve resources has hindered putting into practice simple policies such as *shortest job first* (SJF) or *longest job first* (LJF) o even backfilling, already implemented in LRMS such as Condor.

Therefore, during the last two decades diverse GS implementations have been proposed, but few GSs have survived, not because of their capacity for complex scheduling but because of some other aspects, such as their adaptation to the grid characteristics, scalability or ease of use. It is out of the scope of this thesis to perform an extensive comparison among these GS and the reasons for which most of them were discontinued. Different surveys and taxonomies [133, 134] classify these solutions and tackle their related issues.

For this reason, only Condor-G, WMS and GridWay are described in this subsection, putting more attention on GridWay because it is the system selected for being adapted in this thesis. Thus the aspects detailed through the following paragraphs are needed to understand the whole work performed.

Condor-G and WMS

Currently, the GSs based on Condor [7] are the predominant ones in the largest grid infrastructures in its two versions: Condor-G [8] on OSG and gLite/UMD WMS [9] on EGI. The main difference between both flavours is their user interface, which will determine their final performance and adaptability. Condor-G is used as any LRMS with the local CLI inherited from Condor. Thus, users must login into the machine running Condor-G to submit their jobs, but also they can straightforwardly execute DRMAA applications. In contrast, the approach of WMS is to build a brokering service within the SOA model. Therefore users can only remotely submit and monitor their jobs through its proprietary interface. For this purpose, users usually describe their jobs within files following the proprietary Job Description Language (JDL) and posteriorly, they make use of middleware commands for submitting and controlling these jobs. Consequently, legacy standardised applications are not compatible with WMS.

On the other hand, Condor has some known drawbacks, i.e. it is hard to install, maintain and customise, even the WMS version, and it has some rigidity on the grid because it was initially designed as LRMS. As consequence, these GSs are usually overloaded by users because they are deployed as central services. Additionally, user can not modify the scheduling configuration of a shared service for certain calculations. Moreover, Condor was designed to manage resources belonging to a single administrative domain with stable networks. Its adaptation to the dynamism inherent to grid was not completely achieved. Its behaviour is not efficient in many situations, as it has

been demonstrated in diverse studies [135, 136]. These issues have a larger effect on its gLite/UMD flavour, where the overhead imposed by several layers of middleware results in poor job turnaround.

Nonetheless, Condor-G and WMS have demonstrated high stability and scalability, and are able to support hundreds of thousands of jobs every month. These features make possible to improve the global throughput of large VOs: jobs are always executed on resources that match the job requirements but the grid-wide load balance is maintained. A consequence is that the fault detection, recovery, migration and unattended retries are not completely supported because it is not necessary to achieve this objective. For example, the use of Condor-G is almost limited to the CMS or ATLAS production and to support certain OSG VOs devoted to large computational challenges. WMS is extensively used as provisioning mechanism for pilot jobs in EGI, as will be explained in Section 2.5.

The meta-scheduler GridWay

GridWay[12] [137] enables large-scale, reliable and efficient sharing of the heterogeneous computing resources offered by grid infrastructures via the current UMD/gLite, Globus and other middleware releases. It performs a reliable and unattended execution of jobs transparently to application programmers and end-users. Furthermore, it fully accomplishes the steps [101] described for the Job Scheduling field in Subsection 2.3.3. For this purpose, the design of GridWay is based on the Core element, which orchestrates the different modules and where the characterisation of providers and jobs are compiled (Host and Job Pools); on several user interfaces (CLI, DRMAA, OGSA-BES, and WS-GRAM); and, a set of middleware access drivers (MADs) that allow interfacing with the IS and providers belonging to different infrastructures. In this sense, the providers are discovered and monitored through the drivers handled by the Information Manager (IM) and compiled in the Host Pool. Thus, any job specified through any user interface will be queued in the Job Pool to be subsequently scheduled among the suitable providers listed in the Host Pool. The Scheduler module implements the effective polices that allow this task. For example, priorities can be set for users, groups and providers (i.e. fixed and resource priorities, FP and RP). The fair-share (SH) works together with deadlines (DL) and the current wasted time of jobs waiting in the Job Pool (waiting-time, WT). Moreover, reliability policies are based on accounting the failure rate (FR) of every provider. In addition, the compiled usage (UG) statistics allows prioritising providers following a certain measurement of their achieved QoS or performance for certain application. Additionally, Scheduler takes account of the requirements set for every job, and it ranks (RA) and limits the submission to constrained resources. These scheduling options are summarised in Table 2.1.

Consequently, when a provider is selected to execute a job, the job execution is performed in three steps: *prolog*, for creating the remote experiment directory and transferring the executable and input files; *wrapper* for executing the actual job and obtaining its exit code; and *epilog* for transferring back output files and cleaning tasks. The *prolog* and *epilog* phases are done by interacting with the grid file transfer services (usually GASS or GridFTP), while the *wrapper* step interfaces the grid execution services (such as GRAM2/4/5, CREAM, ARC or OGSA-BES). The architecture of GridWay and the interaction with the grid services of the target infrastructure are

[12] www.gridway.org

shown in Figure 2.3.

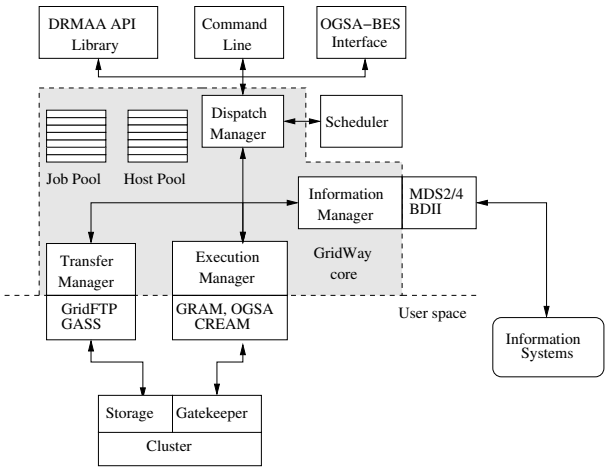


Figure 2.3: GridWay architecture and its interaction with grid services.

Although GridWay efficiently implements some capabilities such as the error recovery mechanism, job migration, and checkpointing triggering, it gives application programmers the responsibility of determining the correctness of the generated outputs, the detection of performance losses, and the application ability to perform the checkpoint. These verifications and mechanisms are considered by GridWay developers out of the scope of a general multi-purpose meta-scheduler, since exit code can be itself the final output of an application. Additionally, GridWay does not automatically deploy monitoring tools for the jobs beyond checking their execution state. This is so because only programmers can properly estimate the execution performance of their applications. In this sense, it only includes a simple trigger in the *wrapper* script that enables the automatically migration or the checkpointing of a job if the CPU utilisation falls down under certain threshold percentage. Users have to modify it to monitor the performance aspect of their interest.

In contrast to the differentiation between Condor-G and WMS, the GridWay framework works on single or multiuser mode, and it can be equally accessed using its CLI, or remotely through its standard OGSA-BES interface [138] and the WS-GRAM [139] one, maintaining its features. However, its deployment is oriented towards individual institutions or users and, with the exception of a few medium-sized infrastructure projects such as GARUDA [140], it has not been selected as part of the production manager systems (PMS) of large VOs; consequently, its suitability for these communities has yet to be demonstrated.

Therefore, GridWay is more oriented to the needs of specialised users and developers than Condor-G and WMS, allowing the creation of specific policies for a given calculation. It takes care of all the necessary steps for the remote execution of a job and is able to analyse the performance of the remote resource. This makes it more reactive [137, 135] to changes in resources, and consequently, it achieves better performance and reliability [141].

Table 2.1: Scheduling options for GridWay configuration.

Scheduler loop options:	
$SCHEDULING\_INTERVAL$	Interval to perform a new scheduling of pending jobs
$DISPATCH\_CHUNK$	Maximum number of jobs dispatched in every scheduling interval
$MAX\_RUNNING\_RESOURCE$	Maximum number of jobs concurrently submitted to same provider
Dispatch priority of a job (j): $P_j = \sum_i w_i \cdot p_{ij}$, **where w is the weight and p the priority contribution of every** $i = \{FP, SH, WT, DL\}$	
FP_USER,FP_GROUP	Fixed priority per user or group (default 0)
SH_USER,SH_GROUP	Ratio of submissions of a user or group over the rest (default 1)
SH_WINDOW_SIZE	Timeframe over which user submissions are evaluated (in days)
SH_WINDOW_DEPTH	Number of frames (present frames are most relevant)
DL_HALF	When pilot or task should get half of the maximum priority assigned by this policy (in days)
Suitable priority of a provider (h): $P_h = f \cdot \sum_i w_i \cdot p_{ih}$, **where w is the weight and p the priority contribution of every** $i = \{RP, RA, UG\}$**. f is 1 when provider h is not banned.**	
RP_HOST, RP_IM	Fixed priority per every provider discovered by an IM
FR_MAX_BANNED	$T_\infty \cdot (1 - e^{\Delta t / C})$, where T_∞ is the maximum time that a provider can be banned, Δt is the time since last failure, and C is a constant that determines how fast the T_∞ limit is reached
FR_BANNED_C	The value of the C constant in the above equation
$UG\_HISTORY\_WINDOW$	Timeframe over which compile execution times to compute their average T_h^{avg} (in days)
$UG\_HISTORY\_RATIO$	The value of the R constant in the equation to estimate the execution time of any job in h: $T_h = (1 - R) \cdot T_h^{avg} + R \cdot T_h^{last}$.

2.4.2.2. IaaS cloud brokers

To be considered a cloud *broker*, a system must perform at least: the dynamic discovering and monitoring of cloud providers; the selection of every provider based on a set of requirements; and, the automatic management of provisioned VMs.

In this sense, based on the OCCI standard, several cloud brokering solutions are being developed nowadays [29, 142, 143, 144]. However, many of them work on a multi-cloud environment that does not allow the automatic discovery of providers. Additionally, it must be clarified when the IaaS providers are statically ranked or selected in a simpler way such as round-robin, without taking into account neither of requirements set for diverse applications nor of the changes in IaaS statuses, the selection performed cannot be considered really brokering or scheduling either. In this work, these tools are denoted as *suppliers*. They repeatedly appear through the related work [113, 30, 42, 43, 145, 146, 147, 114, 115, 31].

However, due to the limited number of IaaS providers, *brokers* were successfully fulfilled by *suppliers* through years. Additionally, VIMs can act as a kind of *broker* when making use of external providers to grow their own resources. To be stackable, VIMs must expose an IaaS interface such as OCCI or AWS. Nevertheless, these protocols are not directly usable by legacy HTC applications [47]. For this purpose, a hybrid API such as DRMAA-OCCI [142] was proposed to enable a direct execution

of jobs on the VMs managed by a cloud *broker*.

With the appearing of first cloud federations, more specialised systems were deployed, but they cannot be considered as VIMs. For example, following the same idea of direct execution, the PMES *broker* [29] offers the OGSA-BES interface to allow instantiating a single VM per each computational task formatted with the JSDL specification. In any case, this type of approaches is limited to the execution of long jobs that compensate for the VM instantiating time.

However, the majority of new approaches are oriented to consolidate complex services on demand, not to offer compatibility with legacy applications. For example, SlipStream[13] was tested in the Helix-Nebula project, although it can work in a multi-cloud environment. It performs VM image management and contextualisation, with virtual cluster automated deployments. It makes use of CPU/Disk/Memory metrics in their scheduling, but it does not inspect the requirements of applications. In contrast, QBROKAGE [143] is an interesting solution recently proposed that focuses the brokering mechanism on satisfying the QoS requirements of applications. Other solutions overcome the absence of federated services building their own ones. For example, CompatibleOne[14] [144] is a complete platform, with its own user management, accounting and monitoring systems. Nevertheless, neither exposes an OCCI service, nor standard interfaces to execute jobs.

2.4.3. Workflow managers

The industrial and scientific needs of counting on graphical tools that control complex production processes, which are described through directed graphs, were the origin [148] of WfMs. Many approaches have been proposed focused on HTC, description of which is out of the scope of this thesis. Different taxonomies and comparatives can be found in [15, 106, 149] and a useful summarisation of the current workflow technologies is schematised in the first figure in [150].

Currently, interactive WfMs such as Kepler [151], Pegasus [152], P-GRADE [60], Taverna [153] or Triana [154] are customary used. Researchers select one of these WfMs according to their features, not only to their suitability for certain calculation type or the support of provenance. In this sense, WfMs are usually selected for their capacity of interoperating among diverse infrastructures (clusters, grids and clouds), or the availability of modules that support the access to specific data sources, such as public databases or commercial cloud storage. For example, Kepler or P-Grade implements adaptors for the different interfaces described in Subsection 2.2.1, and Taverna supports a wide range of public bioinformatic resources based on web services.

The simplest behaviour supported by the interactive WfMs is the step-by-step accomplishment of the work depicted in vertexes, preserving the precedence established with the edges of the graph. This implies time wasted in local tasks or remote data access that is not taken into account in a global scheduling. On the other hand, if some of these expensive tasks are overlapped with the execution of jobs, the scheduling possibilities increase. In this sense, *early dead-line first* (EDF) or *critical job first* (CJF) policies can be easily implemented. However, if the WfM sequentially submits jobs to a shared RB that does not support deadlines or user-defined priorities, there are not guarantees of the critical jobs will be firstly executed. One solution is to inform RB about the whole workflow. For this purpose, languages mentioned in Subsection 2.3.6

[13] http://sixsq.com/products/slipstream.html
[14] http://www.compatibleone.org

should be used. Nevertheless, current RBs (Condor-G, WMS, GridWay) only support simple DAGs [7]. This specification is not complete and even can be considered as obsolete [155] due to it does not take into account communication dependencies or the differentiation among types of jobs. In any case, multiple optimisations can be performed [76] only based on DAGs.

WfMs running in non-interactive mode [156] as well of workflow engines (WfEs) such as ASKALON [157], MOTEUR [158], K-Wf [128], BPEL or Shark support diverse languages that properly enable the characterisation [159] of the workflow and the provenance. The characterisation of the calculation is indispensable to perform an efficient scheduling. Unlike RBs, this jointly features their capacity of working on the Job Building field, allowing these Engines to perform the optimisation of the workflow previously to its execution. The generic phases that transform the abstract workflow specified by the user to the one optimised and its execution are described in [160]. Within these phases multiple techniques can be used. For example the generic horizontal and vertical clustering, which includes the task grouping [161]; Min-Min, Min-Max and other types of partitioning; heuristics as *heterogeneous early-finish-time* (HEFT) [162, 163] or the *localised* HTEF (L-HEFT) [155]); data pre-allocation; or even the efficient distribution [164] based on the characterisation of specific aspects of cloud infrastructures, such as QoS [165] or costs [166].

However, the multiplicity of languages has become an important issue [79] for the knowledge reutilisation and the provenance, in particular for the reproducibility. For example, if the support of certain WfE is discontinued, the results obtained with this system can not be posteriorly verified. In the same way, workflows can not be shared among WfMs. To solve these problems, the Workflow Management Research Group from OGF, the OASIS WS-BPEL Technical Committee, or the Workflow Management Coalition (WfMC) standardisation groups were created, as well as projects such as the SHIWA[15] [150]. Consequently, inter-brokering tools among WfEs, translators and several languages such as YAML, WSBPEL, XPDL or IWIR have been proposed as solutions, but currently the workflow interoperability is far away from being achieved.

2.4.4. Self-schedulers

Due to the impossibility of performing an adaptive scheduling for specific applications based on GSs, WfMs and available middleware, developers have implemented scheduling tools based on their deep knowledge of their applications and the analysis of the infrastructure. This approach is not new and researchers have extensively explored self-scheduling techniques during last two decades. A wide compilation of the algorithms proposed for self-scheduling and their scope was already depicted in Subsections 2.3.3 and 2.4.1, as well as in the last Subsection 2.4.3, because self-schedulers can manage workflows [167]. However, the methods of estimation take special importance due to the unfeasible characterisation provided by the IS deployed in the infrastructures. Therefore, the use of heuristics [168], models [169], as well as the compilation of statistics about aspects that influence on the concrete application [170] are essential to these systems.

Moreover, other scheduling techniques have been used. For example, task replication can not be obviously considered as part of the Workload Division field, although it is probably the most straightforward method to reduce final execution time of MC calculations [171] and even it can improve the parameter sweep ones [172, 173], being

[15] http://www.shiwa-workflow.eu

suitable for accomplishing urgent needs or deadlines. Replication can overcome the needed of characterising the infrastructure because it removes the influence of failed jobs, waiting queues or the hardware performance. Nevertheless, this technique is opposite to Green Computing, wasting many CPU hours in calculations that will finally be discarded and therefore, it should be moderately used. Additionally, recovering from failures on the infrastructure is an issue where there is still room for efficient enhancements. In this sense, algorithms that predict failures can take advantage of checkpointing [174]. In both cases, an issue is to find a balance between resource consumption and desired reliability.

It is noteworthy to mention again that the objective of any self-scheduling framework is to reduce the makespan of a single calculation, composed by one or few applications. In this thesis, only *dynamic* scheduling is considered. Thus ideally, the framework must be able to predict the application consumption and the real availability of resources to *dynamically* adjust the BoT size in a way that allows reducing the overhead generated. Therefore it will support whole *coordination* of the workload (see Subsection 2.3.6), dealing with the scheduling fields described in Subsection 2.3.3, and implementing any other optimisation technique. In general, this is performed by achieving the following phases in a reactive loop:

(1) **Characterisation**: Both infrastructure and application should be characterised. This includes the typical resource discovery in dynamic scheduling, as well as the benchmarking of every resource and the profiling of the application execution. For this purpose, IS are used in first instance to obtain a list of providers, but the benchmarking should be mainly based on real measurements or estimations using the compiled information from point (3).

(2) **Workload Scheduling algorithm**: The Workload Division and Job Building, including resource selection, are performed estimating the turnaround for every possible match among BoTs and providers.

(3) **Submission, monitoring and accounting statistics**: Jobs are submitted to providers following any of the *computation* approaches described in Subsection 2.3.6. Usually RBs are used as *dispatchers*. The execution of those jobs is supervised compiling diverse information. Much of the characterisation procedures of point (1) only can be achieved with the compilation of the final execution times, errors and other aspects with influence on the application performance, which are updated in every loop.

Nevertheless, the suitability of most of the scheduling algorithms proposed have been tested with certain calculations on controlled environments or even using grid and cloud simulators [175, 176, 177], which do not have to adjust to the real behaviour of the production infrastructures. Additionally, the optimisation achieved by self-schedulers is generally based on the deep knowledge of an application or certain calculation, and consequently, the obtained results can not be extrapolated to other fields.

2.4.5. Other tools

Science portals or *science gateways* (SG), such as they are called nowadays, are user interfaces that hide the complexity of the underlying infrastructures and allow researchers to perform calculations by simply pressing a click. In general, they consist

in web pages where the applications are embedded from scratch or directly use specific frameworks such as GridSphere [178] or the Vine Toolkit [179]. In this sense, although their libraries wrap some procedures, the approaches explained in Subsection 2.3.6 are followed for *computation*. Thus, Job Scheduling is usually delegated to RBs. Therefore, in their simplest form, SGs can be considered as applications, or as much, containers of many applications. Examples of them are GENIUS [180] and the current Catania SG [181].

Moreover, there are SGs with more scheduling attributions. For example the ones masking WfEs, such as the e-BioInfra [182] portal, or the WS-PGRADE/gUSE [60], where several users can build and manage multiple workflows. However, the main examples are the production management systems (PMS) of large collaborations, especially those focused on the computation of the data generated by the Large Hadron Collider (LHC[16]). These systems automatically perform advanced scheduling techniques to improve the data allocation as well as support the monitoring of the scientific production thorough long periods of time. Some of them [17, 18, 19] are explained in Subsection 2.5.3 because make use of pilot jobs. In contrast to them, the CMS[17] community has two PMS [16], one for data analysis [183] and one for Monte Carlo production [184], which can submit tasks to Condor-G, gLite/UMD WMS, ARC [54] and some LRMS through a connector called BOSSLite [185].

2.5. Pilot jobs

2.5.1. Overall vision and nomenclature

There is a large collection of literature on distributed master-slave applications for scientific computation. In addition, the definition of a pilot job is commonly extended when the satellite program is being executed on resources belonging to other types of DCIs than grid, such as clouds or even on multiple standalone clusters or volunteer resources. For this reason and because of the wide plethora of topics that have been covered, the systems that have been tested on, or are closely related to, standard grid or cloud environments, are described in this section. This collection is adequate to illustrate the method applied in this study. On the other hand, even though there is a wide range of approaches and possible implementations, any pilot system is conceptually composed of the following main components:

- One (or several) user task queue (UTQ). It is usually a (priority) queue that is accessed by means of a local command line interface (CLI), a web portal, a web service or an API.
- A master coordinator or pilot server (PiS). It is devoted to manage the running pilots, their requests and the task-pilot relationship.
- Pilot jobs (pilots) or agents. In addition to consecutively accomplishing user tasks and enabling remote monitoring of these tasks and the assigned resource, they perform other functions, such as checking and preparing the application environment (downloading and configuring software), executing multiple tasks (multi-task pilot jobs) or accepting tasks belonging to different users (multi-user pilot jobs, MUPJ).

[16] http://public.web.cern.ch/public/en/LHC/LHC-en.html
[17] http://cms.web.cern.ch/news/computing-grid

- One or several pilot generators, suppliers (or *provisioners*), fabrics or factories (PiF). They perform the submission of pilots to a certain computing infrastructure (local clusters, the grid or the cloud). These generators can be manually executed by a user, but they are generally triggered by a daemon that locally monitors the UTQ for pending tasks.

Other additional components that can form part of a pilot system to either facilitate the proper behaviour of the aforementioned modules or improve the performance are the following:

- Message accumulators (MAs) or customised proxies are often deployed to provide a type of bidirectional outbound communication between the PiS and pilots when firewalls or NAT are present.
- The PiS can abandon the passive role and use the support of a scheduler module that implements advanced algorithms to effectively coordinate the pilot-task matchmaking. If only a simple *first come first serve* (FCFS) or a *fit resource first serve* (FRFS) approach is considered for workload scheduling, this role is usually assumed by the PiS.

The implementation of this type of system involves multiple design decisions. The most important of these concerns the method of communication between server and slaves, that is, whether the system adopts a pushing or a pulling behaviour. In the first case, the server can initiate the communication with the slave, e.g., to submit tasks, and in the second case, the slaves initiate all communications with the server. Additionally, the connections can be synchronous or asynchronous, state-less or state-full, and they suppose overheads that the scheduling algorithms cannot overcome. Consequently, these decisions determine the choice of completely implementing the Internet application layer (i.e., directly sending information over sockets), using pure remote procedure call (RPC) libraries (RMI, CORBA), coding embedded methods into web pages (PHP, Python), delegating to external scripts (CGIs), or following a more standardised web services (WS) schema, where the specification of messages has to be selected (XML-RPC, SOAP, WSDL, REST, WSRF), independently from the common protocol (HTTP, SNMP, TCP) used. Furthermore, assuring the security in these communications is necessary because some standard grid middleware will be bypassed. In general, current pilot systems use SSL/TLS authentication and encryption methods based on the user proxy delegated in the WNs, but additional mechanisms may be necessary, especially in cloud, where these proxies are not delegated to VMs. Additionally, a level of isolation in task execution should be considered, especially when tasks are executed on behalf of other users [186].

Regarding all these aspects, a classification of existing design approaches with their corresponding examples is described in the following subsections.

2.5.2. GS/LRMS embedded pilot systems

LRMS and GSs (to a lesser extent) have already incorporated advanced scheduling policies and usually support the interfaces commonly used in HTC (see Subsection 2.3.6). Therefore, it would be a great advantage to include a pilot in their resource pools and schedule tasks over this pilot pool as usual. In this way, ranking and selection of the best resources to meet the task requirements are permitted. This approach was first introduced in grid environments in 2001 by the glidein technique [8], which

was developed using the Condor framework. Subsequently similar mechanisms were incorporated into other systems, especially when cloud providers appeared.

GlideinWMS.

Glidein uses the *condor_startd* daemon, executed inside any computational node and then monitored and managed by Condor using *condor_sched*. This binary file can be wrapped in a script and submitted as a grid job to a remote resource; then, the daemon connects to the Condor *Collector* and it is enrolled in the Condor resource pool. This technique is not suitable for being used by itself in production grid infrastructures, and, for this reason, glidein was recently complemented with other developments [187], resulting in glideinWMS [24].

Currently, this framework is being used to provision resources to high-level systems, such as workflow managers, e.g., Pegasus [188] (through Corral [189]), or it is used directly through the Condor CLI by users allowed accessing core services in OSG [190]. The main differences from a pure glidein system are an enhancement of the grid job wrapper for *condor_startd* and the inclusion of a specialised PiF compound of two principal modules: the *VO* (or *Corral*) *Frontend* and the *glidein Factory*. Multiple instances of *Frontends*, *Factories* and their related software can be deployed to provide load balancing or to access multiple VOs. In this sense, specific *Factories* for cloud provisioning are implemented [113].

Alternatively, the powerful Classified Advertisements (ClassAds) language is used for describing users, tasks, glideins and machines containing services, and, consequently, it enables the Condor matching mechanism by means of the evaluation and comparison of the attributes, expressions, preferences and constraints declared for each actor. However, ClassAds has perhaps derived in a tool that is too complex for conventional users. Additionally, to assure confidentiality, the system generates key pairs for each ClassAd, even for the user's data stored at web servers, increasing the complexity.

The obvious drawbacks of this system are the need for bidirectional outbound connectivity, the compatibility of the *condor_startd* binary, the difficulty of installation and the customisation (inherited from Condor and the newly implemented modules). The first drawback is overcome by using a general-purpose message accumulator (the *Generic Connection Brokering*) or the specifically designed *Condor Connection Brokering* to enable the communication between the glideins and the *condor_sched*, although this approach creates a delay penalty because compiling messages in an intermediate server cannot offer the same performance as LAN connectivity. The second drawback is usually masked by the similar configuration of the resources in the large infrastructures. However, the latter two issues remain important drawbacks to the general adoption of glideinWMS because installing a complete instance could be daunting and difficult for a specific application. Therefore, many users that launch their own applications cannot profit from the scheduling features provided by this system.

Elastic virtual sites and clusters.

Independently of how the IaaS cloud is accessed (with *suppliers*, *brokers* or directly), when the issue of the VM creation is solved, the following main problem is to offer the new provisioned cloud resources in a way compatible with the usual execution of HTC applications. Therefore, many earlier approaches were based on the

set-up of virtual clusters with some specific services or even with a complete virtual grid site. The simplest mechanism is to create virtual nodes at the cloud provider to dynamically grow a private cluster [145, 191, 147, 146]. Other solution is to increase the nodes of a local grid site [42] or even completely place the site at the cloud provider [43]. In such a case, the computational tasks can be directly scheduled by the LRMS of the cluster, or by any grid scheduler, respectively.

The placement of cluster nodes in remote locations has multiple drawbacks anyway. First, the high latency of WANs will prevent the normal behaviour of the cluster and the achievement of some calculations. Besides the unfeasibility of these networks to achieve tightly-coupled parallel applications, the LRMS usually requires shared file systems that dramatically decrease their performance in these environments. Therefore, data and software locality is a great obstacle. Second, the necessity of bi-directional communication among nodes and master implies the assignation of public IPs for every node, the use of message accumulators (MAs) or even setting up a VPN among nodes. IPv4 pubic addressing is limited in cloud providers and can have an associated cost. In addition, security issues must be considered. The use of MAs or VPNs increases the latency of communications. On the other hand, to set several complete virtualised grid sites implies the configuration of much middleware and their subsequent management.

It is important to mention that several tools for automation and monitoring of deployments are currently available for clouds [192], as well as several cloud *brokers* perform similar procedures, and even VIMs such as OpenStack or OpenNebula can be stacked to grow their resources [193]. However, although the work needed to set up a virtual cluster or grid site has been reduced, the performance loss and the complexity acquired, respectively, cannot be justified when other solutions are available.

Managing VMs through SSH.

Other possibility is making use of the contextualisation facilities to directly manage VMs as pilots through SSH commands and to perform the data staging through SFTP. For example, ServiceSs [29] is an extension of COMPSs scheduler that makes uses of the PMES *broker* to provision this type of VMs. The final intention is proposing the system as a kind of platform-as-a-service (PaaS) cloud [194], focused on offering an IDE (an Eclipse plug-in) and a new API based on code directives for the automatic parallelization and orchestration of applications and services in cloud. The approach is powerful and extensible, but not standardised, and legacy applications will be incompatible with it.

Other example is the Service Manager [30], a cloud *supplier* that includes the monitored VMs in the GridWay Host Pool. Subsequently, GridWay can schedule tasks among these VMs and execute them when it makes use of its SSH driver.

Considerations.

Obviously, the main drawback of any pushing mechanism as the ones described in this subsection is the need of direct accessing pilot services from the PiS. In general, this approach implies added costs and overheads. Moreover, the creation of virtual environments such as virtual clusters has not necessary entail a decrease in the complexity. Besides the configuration issues, advanced Workload Scheduling algorithms require the location of the appropriated resources to correctly manage the computation.

2.5.3. Pilot systems related to LHC VOs

Condor-G is used directly in OSG and is the base of gLite/UMD WMS. All previous versions have been deployed on EGI and its preceding phases since the beginning of the last decade. The glidein technology seems the obvious choice to introduce pilot jobs into large infrastructures, especially for those devoted to grid computation from the LHC[18]. However, this idea was only evaluated in recent years. As a result, on the date of the debut of glideinWMS [24], three of the VO communities out of the four main LHC experiments had already implemented their own pilot job techniques and had integrated them into their legacy and centralised PMS for massive data production and processing. These three were DIRAC [17] for LHCb, AliEn [18] for ALICE, and PanDA [19] for ATLAS.

Moreover, the uncertainty of achieving better performance results or getting some added value, compared to their own mature production systems, became a significant drawback for glideinWMS. Since then, its adoption by LHC VOs has been irregular, also because of the influence of OSG on each specific collaboration (mainly on CMS and ATLAS). In this sense, unlike the other LHC collaborations, CMS has not implemented their own pilot system and simply incorporated glideinWMS during the last years. With this system, great results were obtained [195]. However, the use of pilot jobs has played a collateral role and has been bound to OSG sites [196] because of the policy of CMS collaboration. Efforts are being made to incorporate glideinWMS, such that the whole CMS production is expected to pass through this technology very soon.

DIRAC.

The DIRAC WMS [17] was the first pilot system to be deployed in production on a large VO [197], and it continues to use the gLite/UMD WMS as the tool for submitting its pilots to the grid. On the other hand, DIRAC (its VMDIRAC extension) is also the best exponent of profiting cloud infrastructures in production with pilot jobs nowadays.

The pilots (*pilot agents*) install DIRAC (and the VO software) from repositories if it has not already been configured by VO managers in grid sites. They can adopt a *general* or a *specific* role if configured to run either any task or to only accept tasks from a certain user, calculation or specific requirement. This association is possible because of a double-matching mechanism against the UTQ items based on the ClassAd language. Thus, DIRAC can request gLite/UMD WMS the information about resources that accomplish these requirements, filter them according to previous executions and finally submit the pilots. This labour is performed by the *Pilot Director Agent*, working as PiF. When these pilots contact with the server directly fetch the first waiting task in the corresponding UTQ. Therefore, the system is following a FRFS policy where the user fair-share is implemented to a small extent.

To perform cloud provisioning, a new *VM Director* is used instead of the conventional *Pilot Director*. Thus, the architecture of DIRAC is maintained. Currently *VM Director* is a cloud *supplier* that simply dispatches VMs to a pool of known providers [115]. However, it is expected that, following the DIRAC design, the double-matching mechanism among requirements of tasks and the discovery of cloud resources will be applied as they were used with grid sites. Moreover, its functionalities are ex-

[18] http://wlcg-public.web.cern.ch

tended [198] with the advantages of the different types of contextualisation that allow the specific configuration for physicists (HEPiX [199]) or external monitoring (e.g. Ganglia [200]). However, pilots have software dependencies that make users not really being able to choice among virtual environments.

It is noteworthy to mention that DIRAC is deeply described in Section 7.3, due to it is used as one of the pilot systems to be compared with the new framework proposed in this thesis, on a real grid environment.

PanDA.

A similar two-level scheduling concept [190] is proposed by the PanDA pilot framework [19], but based on Condor-G and datasets. Several PiFs (*AutoPilot* [201]) are installed with Condor-G when it is deployed at central services to adjust the number of pilots and tasks in such a way that computing elements (CEs) will not be overloaded. This schedule is based on the input dataset of tasks and their expected output. This is important because pilots have multitasking capabilities and can retain outputs from previous executions to be reutilised or uploaded at a later time.

AliEn.

In 2004, a completely different pilot approach [25] was implemented for the AliEn framework [18]. A daemon (*Computer Agent*) is installed in VO-Boxes at each grid site to continuously monitor the state of its corresponding CE. It also has a PiF role and submits pilots directly to the CE [202] when it detects that tasks have been assigned at UTQ to that resource. Central AliEn services schedule these tasks using the information provided by *Computer Agents*; therefore, the *Computer Agent* (and not the pilot) is triggering the task matching using a general characterisation of the resource. That is, the framework implements more advanced scheduling policies without using the central IS of the infrastructure, avoiding a high percentage of monitoring communications, but at the cost of certainty of a proper WN and the ability to control the task behaviour. In this way, an approach based on site-allocated PiFs was also proposed in DIRAC [203], but the utilisation was limited to standalone clusters [204]. Similarly, some PanDA developers [205] considered the creation of a new site-allocated PiF that supplied glideins [206], but this solution did not seem to be feasible because it forced an unnecessary re-implementation of glideinWMS functionalities and the infrastructure was overloaded with two pilot systems [190].

Considerations.

Notably, these three PMS act not only as a pilot system but also as a complete tool that provides more services. They are frameworks specifically designed to compute the great amount of data generated by the LHC experiments, and therefore, they are composed of additional software mainly devoted to managing data access, allocation and replication. The requirements of tasks processed are specific and they could only accept regular user tasks collaterally. Consequently, PanDa and AliEn frameworks only support High Energy Physics (HEP) experiments. The exception is DIRAC, which is used in other types of initiatives, such as the *biomed* VO [207].

With respect to the communication mechanism utilised in the aforementioned frameworks, all actors (user, CLI tools, pilots, core modules) in DIRAC, AliEn or PanDa

must use their own PKI/GSI protocol based on XML-RPC/HTTP [208], WSRF/-SOAP or simple HTTP requests, respectively. For this reason, in addition to the web portals offered by PMS, more common interfaces are provided to developers to facilitate the integration with other systems or applications, although they are not necessarily standardised. Examples are the Ganga [121] compatibility that is usually offered or the REST interface recently added to DIRAC. With respect to the coding language, Python is predominantly used. Additionally third-party open source tools such as web servers are used.

2.5.4. Application-oriented overlays

HEP researchers involved in the four LHC experiments generally use the same software for processing their data; therefore, it is feasible to make the effort to customise a common framework for them. Nevertheless, there are many users that employ several types of applications not installed on remote sites. Thus, the development of a PMS only makes sense for middle-large collaborations. Moreover, the deployment of any type of centralised system for queuing and prioritising tasks may not be the best approach to improve the performance of an individual application because only its programmer knows its specific characteristics. For this reason, some approaches have been proposed for providing a developer-oriented interface for distributed environments, which tackles the implementation of master-worker applications. Initially they focused on offering an API that simplifies the basic operations of the slave (monitoring, forking tasks, getting results, etc.). Two similar approaches were proposed: AMWAT [209] for AppLeS [210] scheduler and M/W [211] for Condor. The former is a collection of C/C++/Fortran functions and the latter implements a set of C++ template classes that the programmer must customise for his application. Both APIs simply wrap the available message passing protocols (MPI, PVM, or Condor-PVM) or the file sharing (Condor-Local I/O, Globus I/O or Nexus) mechanisms to enable bidirectional communications in a distributed environment.

BigJob

Due to the high latency in wide area networks, these protocols are not suitable for grid environments. For this reason, other approaches based on GridRPC [212] were attempted [213], but all of these were lacking adoption when standard APIs for HTC programming were proposed (DRMAA [214], SAGA [120]), which did not consider a role for the programmer to manage the slaves. Related to this, an initiative called BigJob [26] has been recently presented for adapting pilot jobs to SAGA, but characterisation or scheduling tools are not included. Developers must explicitly indicate the pilot job to execute a task, and usually indicate the provider (its URI) to run the pilot too. Thus, the *BJ-Manager* assumes the UTQ, the PiS and (partially) the PiF roles to facilitate developers the management of pilots (the *BJ-Agents*). The objective is to maintain the freedom to implement any provisioning policy, but without the necessity of directly implement SAGA code. However, due to the volume of resources belonging to current grid infrastructures, provisioning is usually delegated to Condor-G. Another component of BigJob is the *Coordination Service* which acts as a message accumulator between the *Manager* and the *Agents* based on the Redis[19] database and protocol. This communication mechanism is always used although other methods

[19] http://redis.io

Table 2.2: Comparison of some distinguishing features of main pilot systems. This compilation is focused on their compatibility and their capacity for interoperation in grids; their tools to adapt applications and their management by users; their default scheduling capacities; and how users can influence the scheduling.

	GS/LRMS embedded		**PMS**			**Application-oriented**	
	glideinWMS	GWpilot	DIRAC	AliEn	PanDA	DIANE	BigJob
Deploy-ability:							
Pilot requirements	Compiled binary (*condor_startd*)	Python >= 2,4,3	Specific Python release and multiple binary dependences	Executable locally stored in shared directory (VO-boxes) or proxy-cache (CVMFS)	Python 2 (only if generic pilot is deployed, but specific versions and middleware were need)	omniORB libraries	Python 2.7
Connectivity (Pull/push)	TCP sockets (Push. MAs are need: GCB/CCB)	HTTP/S (Pull)	DISET, i.e. XML-RPC/HTTP (Pull)	WSRF/SOAP (Pull)	HTTP/S (Pull)	CORBA call-backs (Pull)	Redis (Push. *BJ-Manager* uses the *Coordination Service* as MA)
Grid interoperation for provisioning[(a)]	Integrated with Condor-G: direct submission to GRAM2/4/5, CREAM and ARC resources (mainly based in middleware libraries)	Integrated with GridWay: direct submission to GRAM2/4/5, CREAM, OGSA-BES and ARC resources (mainly based in middleware libraries)	Submission delegated to WMS. Additionally, allows direct submission to statically defined GRAM2 and CREAM resources (based on middleware commands)	Site-allocated PiFs perform local submissions to GRAM2, CREAM, or ARC resources (based on middleware commands)	Submission delegated to Condor-G	Relies on Ganga: submission delegated to WMS, PanDA and DIRAC or directly to GRAM2, CREAM (based on middleware commands). SAGA backend (deprecated)	Relies on SAGA-python and SSH: submission delegated to Condor-G (potentially can be extended or use other SAGA implementations to handle grid interfaces)
Usability / Interactivity:							
Local CLI	LRMS-like	LRMS-like	-	-	-	Limited	Limited
Remote CLI	-	SAM based (through OGSA-BES)	WMS-like (extended)	UNIX-like prompt	Few parameterised commands	-	-
Task description (for CLI)	Proprietary	JSDL, proprietary	JDL	JDL	-	-	-

(Continue in next page.)

(Table 2.2 continued.)

Task management API	Proprietary (Perl, SOAP). DRMAA (C/C++)	DRMAA (Java, C/C++, Perl, Python), OGSA-BES	Ganga. REST. Proprietary (Python)	Proprietary (C/C++/Perl)	Ganga, REST	Proprietary (Python)	Proprietary (Python)
Scheduling:							
Default workload policies	Condor matchmaking	GridWay adaptive scheduling	Classification of tasks in UTQs, subsequently FRFS	Tasks in a priority UTQ are assigned to sites if their local monitoring shows free slots and they fulfil constraints	Classification of tasks based on their datasets. Subsequently FRFS	FCFS	Developer must explicitly indicate the pilot to execute tasks
User defined policies	Free definition of constraints or ranking	Free definition of constraints or ranking	Few types of constraints	Constraints related to software installed	Constraints related to software, input files or sites	Free definition of constraints. It allows dynamically setting scheduling options	-
Provisioning policies	PiF dynamically evaluates policies defined in tasks and static options to generate pilot descriptions. Subsequently, Condor performs their scheduling and submission	PiF dynamically evaluates policies defined in tasks and static options to generate pilot descriptions. Subsequently, GridWay performs their scheduling and submission	For every UTQ, PiF select resources from a list provided statically or dynamically by WMS. Subsequently resources are filtered by previous pilot profiling and static options	PiF discretionally submits pilots if tasks are assigned to its hosting site at central UTQ	Several sites are selected to send the inputs of tasks with similar datasets. Then, PiFs discretionally submits pilots to these sites using Condor-G	Only a type of resource can be selected. Scheduling is generally delegated. If WMS is used, PiF can also perform a limited scheduling based on previous executions	Developer must explicitly indicate the resource to execute pilots (if it is not delegated to Condor-G)
Pilot characterisation [(b)]	ClassAds (Not customisable)	Unstructured key-value pairs (Customisable)	ClassAds (Not customisable)	ClassAds (Not customisable)	Key-data structure pairs (Not customisable)	Key-data structure pairs (Limited[(b)])	Key-data structure pairs (Not customisable)

(*a*) SSH or cloud interfaces are not considered as grid interfaces. Accessing to LRMS is only considered if the motivation is to bind with grid interfaces, as in Condor-G.
(*b*) Characterisation by users without modifying pilot code. Special case is DIANE, since *Worker Agent* could be considered part of the user application, due to CORBA model.

are available, for example when pilots are started through SSH in VMs previously provisioned in cloud [114].

DIANE

Unlike BigJob, DIANE [23] is an integrated framework with a UTQ and simple scheduler (*Task Scheduler*), PiF (*Agent Factory*) and PiS (*Run Master*) that is written as an importable API library for Python applications. The communication between pilot jobs (*worker agents*) and the PiS is done by means of the CORBA free implementation omniORB[20]. However, it maintains a pull behaviour, where the exchange of information is done via call-backs. The *Agent Factory* can use simple heuristics [68] to fit resources by evaluating its error completion rate. Nevertheless, the resource discovering is based on the statistical results obtained from the pilot submission to the Ganga [121] library to a set of GSs, as is deeply explained in [68]. The *Task Scheduler* implements a FCFS policy by default, although its code can be modified to incorporate other procedures. Additionally, the *Agent Factory* and the *Task Scheduler* can work together to exchange information about the characterisation of resources and tasks. Therefore, specialised developers can extend DIANE to allow ranking policies and improve the resource provisioning with statistical data. The *adaptive workload balancing* (AWLB) proposal [117, 215] is a good example.

Considerations.

The primary advantage with DIANE or BigJob is their library design, which provides an easy-to-use and standalone installation. In the case of DIANE there is an added benefit because developers only have to divide the computational workload into tasks and to format them into a set of simple abstract classes. Then, DIANE will autonomously manage the necessary pilots and, through its own *File Transfer service*, will make the stage-in and -out process. However, both APIs are not standardised, and they even differ from the Ganga interface or SAGA standard, respectively. Moreover, the programmer could be forced to modify the DIANE [216] or the BigJob code if the existing one did not meet his needs. The performance data and characteristics of resources can only be compiled for the current execution of the application; therefore, this information can be shared neither with other users nor with other applications of the same user that are being executed at the same time.

2.5.5. Other frameworks

In general, desktop grid approaches are similar to standalone pilot systems, but they have been logically designed for Volunteer Computing. Therefore, they were focused on exploiting idle CPU cycles from a large number of personal computers variably configured and allocated around the world. This fact implies a great effort in the development of compatible and secure *Clients*, which can run on heterogeneous platforms, and also centralised services able to coordinate hundreds of thousands of *Clients* while supporting high latency rates in their communications. In the case of BOINC [217], significant segments of the *Clients* and the core server are implemented in C++, in particular the communication that relies on XML/HTTP(S) requests from the *Clients*. The rest of BOINC, e.g., the mechanism for downloading files, is based on

[20] http://omniorb.sourceforge.net

PHP web pages. Another example is XtremWeb [218], where *Clients* connect through RMI or XML-RPC to *Coordinators*. The servers are published at an *Advertisement Service*, also coded as an RPC. This enables a type of P2P load-balancing mechanism because *Clients* can obtain inputs from and store outputs to different *Coordinators*.

There have been efforts to adapt XtremWeb and BOINC architectures to a grid infrastructure, such as EDGeS [27] or GridBot [28], but legacy grid applications are difficult to port to these frameworks. Nowadays, the recent 3G-Bridge [31] implementations are able to maintain a BOINC server with *Clients* (the pilots) running in provisioned VMs. In similarity with grid, the work is focused on the customisation and the contextualisation needed to accomplish the common user-tasks in the *Clients*. In this case, a modified CREAM server which relies on 3G-Bridge can be deployed as a facility for grid users. However, the scheduling among diverse cloud sites is relegated, using *suppliers* to single providers.

Other initiatives have tried to solve this issue by implementing systems from scratch that were immersed in the current grid middleware and its interfaces. This is the case of Falkon [219], which was implemented as the Globus Toolkit 4 (GT4), that is, a WS-oriented system to be deployed on existent GT4 infrastructures. According to its initial design, a PiF (*Provisioner*) directly manages the execution of pilots (*Executors*) running on WNs, assuming *a priori* that GT4 supports brokered WS notifications, i.e., the PiS (*Dispatcher*) must push WS notifications directly to *Executors* running on a remote resource; however, this feature has not been implemented in GT4.

Considerations.

Obviously, the library dependences of *Clients* will hinder: the deployment of these systems, the adaptation of legacy applications and even the election of the VM configuration in cloud environments. Furthermore, the technologies added with 3G-Bridge are oriented to improve the compatibility with applications based on the UMD/gLite middleware, but mainly to achieve interoperation among infrastructures, and not to offer the necessary tools for customising scheduling capabilities.

Corollary.

To better understand the concepts described through these subsections, Table 2.2 summarises the distinguishing features of main pilot systems in relation to their deploy-ability, usability and scheduling. The comparison is specially focused on the mechanisms offered to users to adapt their legacy applications to these frameworks, as well as to perform some type of customised monitoring and scheduling. Additionally to this table, it is important to mention that DIRAC and DIANE frameworks are deeper explained in Section 7.3 because they are used to be compared with GWpilot.

2.5.6. Limited support to Multilevel Scheduling

The considerations made in last section suggest that no pilot system fulfils all the needs of the developers, users or institutions. In particular, they do not allow users to modify their scheduling behaviour in a way compatible with legacy applications or tools. However, there are diverse examples of specialised scheduling that make use of those frameworks. In first place, systems that support a formalised execution model [12, 116, 117] are the ones more suitable for incorporating advanced Workload

Scheduling algorithms. In this sense, the improvement of MC codes is recurrent because of their wide use in grid environments and their mouldability. For example, in [220] is shown how DIANE can be used to dynamically balance the MC payload among available pilots, the procedures of which are an adaptation of the general workload balance explained in [216] based on the proportion of transfer time over execution time. In [207], the MapReduce and checkpointing techniques are applied to the DIRAC framework. Additionally, MapReduce has been developed on BigJob [44]. Nevertheless, these examples are also too simplified to be extrapolated to other algorithm types that require more complex characterisations or features. This is the reason of the implementation of proposals that disregard the current frameworks and implement customised ones, such as the inclusion of data-cache pilots [81] in the MC production system of CMS collaboration. Despite of these efforts, any legacy application should be rewritten to use any of the described mechanisms.

A more ambitious approach is the possible inclusion of Moab scheduler into DIRAC framework [221]. It could contribute to improve the generality of this pilot system by including the advanced scheduling algorithms present in LRMS such as fair sharing rules or backfilling. However, Condor-G already provides these advanced policies and its application in glideinWMS is very limited. This is so due to the impossibility of customising the characterisation of pilots. Moreover, currently there are some approaches [145, 191, 147, 193, 192] to guide cloud provisioning to set virtual clusters up, but application developers are unaware of the distributed nature of the virtual environment.

Coupling external systems with scheduling capacities to pilot framework is difficult with this lack of compatibility and characterisation, but some approaches have been developed. For example, a WfM such as Pegasus makes use of glideins to face the former problem. This approach achieves good results [189] with the drawback of the local dependence on Condor, which limits its development. Other option is the one followed by MOTEUR. Their developers have been used the Ganga library to become the system in a pilot-enabled WfE that relies on DIRAC [222]. Both mechanisms allow WfMs to straightforwardly use pilots, but not to characterise them. For this purpose, other authors had again to adapt DIANE API to be used by MOTEUR [223], and then to build a scientific portal [182] on the top of them.

2.6. Conclusions

Firstly, it can be said that there has been too much effort spent worldwide on building advanced Workload Scheduling with early-binding techniques. New developments should take advantage of the tools already implemented. In this sense, technologies progress and the backward compatibility must be preserved. An example is the research on the compatibility of workflow languages for the provenance. However, the most suitable approach is following standardised APIs when the Application-level Scheduling is implemented. In this sense, some algorithms included in self-schedulers or WfMs are so specific that should not be part of a general-purpose scheduling system such as the RBs (or a pilot framework); unlike, they should couple them making use of those standards. Additionally, generic techniques should be tested on real environments before being incorporated into an established framework, and self-schedulers are the first tool for this purpose.

Obviously, the lack of characterisation in early-binding limits the feasibility of

many scheduling approaches, so pilot jobs can be the solution. However, this affirmation is not clear for many authors, who persevere in the research without using Provisioning, as well as for many communities, that continue relying on early-binding mechanisms in their scientific production (such as some PMS). They trust a careful selection of the resources and the size of BoTs to improve the performance in a similar degree than pilot systems, especially when long jobs are used and the global throughput is considered.

This is so because current pilot frameworks are not exploiting all the advantages that the pilot job technique offers. Workload Scheduling and Provisioning do no work together, and they cannot be easily modified for a specific calculation. To enable Application layer to benefit of Multilevel Scheduling, pilot frameworks must allow the:

1. compatibility with legacy applications and Workload Scheduling tools such as WfMs and self-schedulers that use standards to bind to other systems;
2. fully characterisation of resources that results into a simple turnaround model suitable for being used by other scheduling layers, this include:
   a) predictable performance of the system based on its configuration;
   b) dynamic customisation of monitoring, without modifying the code of the framework;
   c) tools to dynamically check the monitored characteristics;
3. communication among layers, transferring requirements, monitoring and guiding from application to Provisioning and coming back.

These requirements are not fully achieved by current pilot systems. It is noteworthy to mention again that some abstraction models of pilot jobs have been already proposed [12, 116, 117]. These abstractions and formalisms undoubtedly result adequate to describe scheduling algorithms based on pilots, but are succinct to detail the design of some frameworks, which are constrained by some issues, such as the compatibility or the deploy-ability, among other weaknesses explained through the Section 2.5. On the other hand, the efficient pilot provisioning in cloud is a practically unexplored field.

Therefore, to design a new pilot framework that will overcome all those issues, it is necessary to detail its requirements, compiling the knowledge acquired through this chapter, as well as to itemise the consecutive sub-objectives of the research, extending the main objectives stated in Section 1.2. These are the reasons for highlighting following Chapter 3.

Chapter 3

Lessons Learned and Objectives of the Research

3.1. Objectives

The main objective of this work is to research into a new framework that will really enable users, institutions and communities to perform the specialised Multilevel Scheduling that fulfils their computational needs. For this purpose, the experience, the tools and in special, the Workload Scheduling algorithms implemented for early-binding should be preserved despite of their limitations. Moreover, some early-binding mechanisms can be extended to be subsequently profited in a late-binding environment. Therefore, these approaches will be explored in the first part of this thesis by means of:

Part I. Chapter 4.: Adapting a consistent set of applications from several (scientific or industrial) areas with distinctive requirements and studying the suitability of different procedures for their adaptation.

Part I. Chapter 5.: Looking into mechanisms that will increase and improve the available amount of resources with the profiting of IaaS cloud providers in a way suitable for accomplishing the previously adapted applications and preserving Workload Scheduling already implemented.

Part I. Chapter 6.: Deploying specialised Workload Scheduling algorithms with self-scheduling techniques and evaluate if they are valid to improve the execution of those applications.

Through the second part, the new pilot framework that accomplishes the requirements for building a Multilevel scheduler will be presented. For this purpose the following steps will be performed:

Part II. Chapter 7.: Designing the new pilot system and comparing its basic features and performance with other solutions by means of competitive experiments.

Part II. Chapter 8.: Creating a consistent mechanism to straightforwardly run legacy applications such as the ones adapted in Chapter 4, for which they can profit from the advantages of Resource Provisioning, even on cloud environments by using the approach developed in Chapter 5.

Part II. Chapter 9.: Implementing new scheduling policies that improve the global performance of several applications at the same time and demonstrating their inclusion does not modify the portability of legacy codes.

Part II. Chapter 10.: Developing the needed techniques to fully enable the scheduling customisation at user-level, allowing the effective communication among scheduling layers.

Part II. Chapter 11.: Conceiving a feasible model suitable for being used by external tools that works on Workload Scheduling, formulating a methodology to incorporate them to the new system and, performing a demonstration by stacking a self-scheduler as the one deployed in Chapter 6.

To accomplish these steps, whole demonstrations must be performed on real environments, i.e. production infrastructures. Moreover, experiments described in the Part II must stress the behaviour of the system when it faces the worst scenario, i.e. dealing with variable short jobs. Both requirements are needed to evaluate the real performance and suitability of the frameworks considered.

Collaterally, other important objective of this thesis is to demonstrate how customary applications can directly benefit from the new pilot framework, providing relevant results for their areas of knowledge. In particular, the fifty percent of calculations are devoted to the study of transport in Nuclear Fusion. For this reason, the related physics and the new results found are extensively explained through the Appendix C of this thesis, while the description and achievements in other fields are summarised in Appendix B.

3.2. Requirements for a new pilot system

The described pilot systems in Section 2.5 have some problems already explained that do not only hinder their support of Multilevel Scheduling, but even can prevent their deployment for simple computations. The orientation of these systems to certain calculation is the reason of going away from a general-purpose and user-oriented Multilevel scheduler. Basing on the related work, there are a set of minimal requirements that a pilot framework must accomplish to be considered as a general-purpose pilot system. While other features finally provides system with the Multilevel capacity.

3.2.1. Minimal functionalities

Friendly interfaces.

First, a general-purpose pilot system should offer a friendly interface for users, developers and administrators. For this reason, a new pilot system must provide

standardised APIs and LRMS-like CLIs that fulfil the needs of every role, and facilitate the incorporation of external systems, such as upper scheduling layers.

Commonly used protocols and grid security.

Security, based on grid standards, is a must. Then, other implementations, different from the ones provided by the middleware, must follow these specifications and general infrastructure rules. New services or protocols should not be implemented if they are correctly provided by the middleware, i.e., the system should avoid duplication; this includes the mechanism to stage files [23, 24], which is already available using GridFTP, SRM, etc.

Deploy-ability.

The platform and library dependencies of remote pilot software [17, 23, 24, 27, 28], i.e. the pilot jobs, decrease the compatibility with heterogeneous resources. Moreover, the constraints imposed by these software make users not really being able to choice among virtual environments in cloud [115, 198, 31]. Therefore, it is better to use a widely extended interpreted language [19, 26] so those future modifications will be easier to be implemented as the system remains widely compatible. In addition, the allocation of modules at remote sites (such as site-allocated PiFs [25, 203, 204, 205]) should be discarded because it implies external collaboration for their installations, which is not suitable for standalone solutions. With respect to the local installation of the pilot system, it is desirable that it were easy enough to be carried out even by inexpert users [23, 26]. This feature will extend the suitability of the system for individual calculations.

Pull approach.

The use of the pull mechanism and common protocols is the most suitable approach for pilot communications. It facilitates passage through site firewalls and proxies. Conventional RPC protocols and bidirectional implementations are not recommended. The use of message accumulators also increases the complexity and lag times in communications [24, 26, 116]. The same reason makes the use of encrypted protocols such as SSH or VPN inadvisable [29, 30]. However, the choice of a pull method can influence the scheduling algorithms selected, as will be explained later. In any case, it is necessary to reduce and maintain controlled the overheads introduced by the system because they could constraint any scheduling decision.

Controlled overheads.

To reduce the overheads, the middleware layers must be limited to the minimum. It is counterproductive to mix different pilot systems (as in [26, 205, 206]) or to complicate the design in excess (as in [17, 24]). A high number of standalone modules in the system does not only imply more difficulties in their deployment, but also multiplies the quantity and size of communications among these modules, increasing the overhead. Well-defined WS message protocols are precisely proposed to easily integrate software from different providers and to improve the future extensibility of codes by other developers [25, 208], in exchange for a high increase of the message size. However, if the system complexity is low and the performance in communications

is a priority requirement, their use could be avoided [19, 28]. Although these simplifications were made, the different pilot calls always introduce their own overhead, which has to be measured and controlled because it has an especial impact on task turnaround.

3.2.2. Multilevel support

In general, pilot systems should distinguish the three-level hierarchy already established for scheduling in Subsection 2.3.1: the Application, the Task Scheduling and the Resource Provisioning layers. To fully support a personalised scheduling at user-level, the new pilot framework must provide the latter two layers with mechanisms that permeate the user's requirements. Thus, these requirements must influence on the whole scheduling because applications properly run on a type of resource that should be provisioned. However, it is difficult to guide Provisioning if the pilot execution is delegated to an external GS [17, 23, 201, 31]. Thus, to build a framework that manages tasks and Provisioning together [24, 26, 116] is more feasible.

Customisable characterisation.

To really solve the characterisation problem of Grid and Cloud Computing, the system must allow the free definition of constraints for every task, among a customisable set of characteristics for every pilot. For this purpose, the language used can not be complex as in [8]. Additionally, the customisation and characterisation of resources is a typical role of the Provisioning phase [17, 19, 25], but will imply the modification of the pilot system code for every calculation type. Therefore, the new pilot system should also provide the mechanisms to properly characterise resources inside tasks. Moreover, applications performing reactive scheduling need interfaces to dynamically know the current task, pilot and resource assignation and characterisation.

Fair-sharing.

To satisfy the needs of institutions and VOs, the system should be multi-user and multi-application. Thus, another desirable functionality is the ability to implement fair-share and prioritisation policies [24] at Workload and Provisioning levels from the start. It becomes a major drawback if they are need post-development actions [221]. It is also useful to have accounting generically provided to the scheduler to improve the behaviour of current and future applications in the system. Therefore, a unique application-oriented approach (such as [23, 26]) is not suitable for designing the new framework.

Advanced scheduling, performance and modelling

In this sense, it is necessary to include advanced algorithms into the Task Scheduling and Provisioning layers to delegate as much as possible the implementation of *coordination* from user-level applications to pilot system. Therefore, the new pilot system must be able to implement more advanced algorithms than FRFS for Task Scheduling. However, this requirement deserves a fuller explanation because is different to the approach implemented in current pilot systems that make use of a pull mechanism. This is so because in a pure pull scenario, the simple FRFS task-pilot matchmaking policy has clear benefits. The technique has a small dispatch time

(below 1 second [23, 204]); it maximises the usage of assigned resources (CPU occupancy); only one call type against PiS should be implemented, and it removes the necessity of implementing a scheduler module. Nevertheless, the disadvantages are also obvious. Although UTQ could prioritise tasks, the user fair-share is not really performed [221]. Additionally, applications can experience large, variable completion times because any task can uncontrollably fall into slow nodes. Then, this policy supposes a large drawback from the user point of view and from the use policy of certain VOs. Nevertheless, any algorithm different from FRFS has a variable time penalty according to its computational complexity, arising from evaluating a large quantity of possible pilot-task matches at every time. This could also have a negative influence on the task dispatching time (when it wastes time on the order of minutes), especially for short-duration tasks, and must be evaluated. In this sense, the execution models available [12, 116, 117] do not take into account these issues and they are tied to the pilot systems for which were formulated. Therefore, it is needed to propose a new mathematical model that accurately compiles these overheads. Such a model must be focused on estimating the task turnaround to allow the fine tuning of the new system as well as the incorporation of third-party schedulers.

Part I

LIMITATIONS AND ADVANTAGES OF EARLY-BINDING TECHNIQUES

Chapter 4

Adapting Applications

4.1. Introduction

Several applications belonging to different areas of knowledge have been adapted to grid and cloud to be studied in this thesis. Tools for Chemical Physics, Evolutionary Biology, HEP, Matter Interactions, Nuclear Fusion and Solid State Physics have been chosen because of their differences on their computational behaviour and to demonstrate the universality of the developments presented in this research.

In this chapter the objective is to evaluate the suitability of the different procedures for their adaptation to highly distributed environments. For this purpose, it is necessary to describe these processes and to perform experiments that measure the performance of the available scheduling tools, in particular RBs because they are the first mechanism to reduce the grid (and cloud) complexity.

Among all those applications, the Drift Kinetic Equation solver for Grids (DKEsG) is excellent to illustrate the usual work on the adaptation of HTC codes to Grid Computing, because it manages extensive parameter sweep calculations as well as complex workflow executions. Furthermore, DKEsG is repeatedly used through this thesis. For these reasons, the complete description of its adaptation has been included in Section 4.4. Results obtained with other applications have been summarised in the following section. Additionally, Section 4.3 describes the approach followed in the adaptation of these codes when DRMAA is used.

4.2. Collection, mechanisms and summary of results

Fifteen applications have been adapted to distributed environments in this thesis. A complete description and their usage context are included in Appendix B, but in this section these applications were classified following the computational suitability types from Subsection 2.3.2, namely: random numbers, parameter sweep and workflows. On the other hand, to perform these adaptations, several approaches were followed making use of:

1. Two RBs available for Grid Computing, WMS and GridWay, through their remote and local CLI, respectively.

2. The standardised DRMAA API. For this purpose, a design pattern has been

newly implemented to ease the adaptation.

3. A Visual WfM such as Taverna [153].
4. A self-scheduling framework, developed in Chapter 6.

Both classifications jointly to the maximum speedup obtained through several published works are summarised in Table 4.1. It is noteworthy to mention that, although some applications have been previously adapted by other authors to diverse grid middleware releases, most of this work is not freely available. Therefore, the process has been done again. Moreover, for most applications, such as ISDEP, the adaptation was not done before relying on standardised API specifications. This achievement has been performed in this thesis with the library described in the following Section 4.3.

However, the speedup does not offer a true vision of the real suitability of the approaches implemented. On the one hand, it already demonstrates the calculation can be distributed, but on the other hand, the scalability and performance are deceptive. For example, the main component of the PhyloGrid workflow is MrBayes. The used version is a tightly-coupled code that must be executed on clusters because requires MPI. Thus, the speedup shows the maximum number of cores reserved in a grid site, and the submission of different experiments to different sites is the only way to perform some distribution. Other examples are DiVoS, jModelTest2 and ProTest3, which do not scale linearly in distributed environments. These codes have an important common portion of calculation that cannot be parallelised and every job must execute it. Amdahl's Law inevitably limits the speedup. Therefore, grids and clouds can be used for small or medium tests, but for large calculations, shared-memory architectures based on GPUs are more suitable for these codes because the sequential part is only executed once. Therefore, these applications were discarded in favour of the rest to perform the experiments analysed through this thesis. That is so because the mouldability property of later ones will allow better exploiting and demonstrating the achievements presented.

Moreover, the dependences and installation size of applications such as XMM-Newton SAS, GAMOS and even FLUKA, hinder their deployment. Their workload is mouldable, but their execution requires the installation of the software in every grid site. For these cases, cloud features can be the solution and for this reason XMM-Newton is used in Chapter 5.

Furthermore, as can be seen in Table 4.1 better results have been obtained with WMS than GridWay in some cases. This seems to contradict previous conclusions made through the related work [135], but it is not so. The status of the infrastructure used or simply fate in the first election of sites influence on the final makespan. Note that IS do not show an accurate characterisation of the providers and GridWay blindly selects providers at first time. This has special incidence when very long jobs are executed. Nevertheless, the probability of failures increases as the duration of jobs does, and many jobs will be re-submitted, wasting excessive time. Therefore, it is necessary an extensive analysis of the carried overheads to evaluate their influence on makespan. With this study, any user can determine if RBs or self-schedulers are enough to achieve the required performance levels for its calculations. For this purpose, the first mechanism in Section 4.5 and both through Chapter 6 are evaluated. In this sense, the experience on Visual WfMs obtained with PhyloGrid shows that it is difficult to customise their scheduling behaviour. Consequently, as it is not possible to easily improve the performance already achieved by RBs or by a self-scheduler, WfMs have not been included in that study.

Table 4.1: Maximum speedup over the sequential execution of the adapted applications obtained in several works using an early-binding approach.

Type	Application	WMS CLI	GridWay CLI	DRMAA pattern	WfM	Self-scheduling
Random numbers	BEAMnrc [67]	3.0[b]	-	17.3[b]	-	28.0[b]
	FAFNER2 [224, 225, 226, 67]	22.9[b]	-	28.9[b]	-	76.0[b]
	FLUKA [67]	22.5[b]	-	10.7[b]	-	31.1[b]
	Grif [67]	83.5[b]	-	37.2[b]	-	79.2[b]
	ISDEP [227]	58.0[b]	64.9[b]	(!)	-	78.3[b]
	Nagano [67]	9.5[b]	-	10.6[b]	-	39.8[b]
Parameter sweep	DiVoS [228, 229]	-	22.0[b]	-	-	-
	DKES [230]	-	29.7[a]	(!)	-	-
	jModelTest2 [231, 232]	-	-	12.9[c]	-	-
	ProTest3 [231, 232]	-	-	16.4[c]	-	-
	XMM-Newton SAS [233]	-	(!!)	29.6[d]	-	-
Workflows	DKEsG [234]	-	17.0[a]	(!)	-	-
	FastDEP [67]	2.3[b]	-	2.1[b]	-	2.8[b]
	PhyloGrid [235, 236, 237, 238]	-	-	-	20.0[a]	-
	GAMOS [239]	-	75.4[a]	(!)	-	-

[a] EELA-2/GISELA (small sized infrastructure, 10-16 grid sites; i.e. *prod.vo.eu-eela.eu* VO).
[b] EGEE-III/EGI (medium sized VO, 40 sites; e.g. *fusion* VO).
[c] EGEE-III/EGI (large sized VO, 120 sites; e.g. *biomed* VO).
[d] EGI FedCloud (medium sized VO, 40 sites, i.e. *fedcloud.egi.eu* VO).
(!) DRMAA version only used with late-binding [240, 241, 65, 242], see Chapters 9, 10 and 12.
(!!) CLI version used on a controlled environment, thus the speedups obtained should be discarded.

4.3. Standardised producer-consumer design pattern

The advantages of adapting distributed codes using libraries and specification standards have been already commented in Section 2.3.6. However, developers do not want to deal with the *computation* part and even with the *coordination* one if they do not plan implementing a specific scheduling algorithm for their calculation. Additionally, SAGA, DRMAA and BES can be embarrassingly, demanding to invest time in learning and training.

These issues have motivated the creation of a library that follows a producer-consumer design pattern and allows the straightforward adaptation of HTC applications with DRMAA. The producer-consumer library is written in Python and, unlike similar tools that run on GridWay [243], the library is potentially compatible with other GSs or LRMS that support DRMAA 1.0, and consequently, makes any distributed application as a standard-enabled software that can straightforwardly run on clusters and grids.

Developer implements only two abstract classes to adapt their calculation with this library: the *input pool*, and the *application*, which is composed by the *producer* and the *consumer* methods:

- The *application* class implements the common procedure to build a single job and verify their outputs. For simplicity, the developer creates a Python dictionary that describes the job as it were a GridWay template within the *producer* method. In every *application* instantiation, *producer* is called with the corresponding parameter combination, building the job. Finally, developer checks the exit code of the job and its outputs with the *consumer* method.
- With the *input pool* class, the developer creates the pool of parameter combinations to be computed, which is automatically stored in a local database. Besides the input parameters, every element of the pool is composed by the object class of the *application* and pointers to other elements to preserve precedence constraints. As several *application* classes can be implemented, the library can manage workflows.

The producer-consumer operation is the following:

1. User launches the application without any concern for the implementation, but he must configure at least the following options with influence on performance and overload: the *polling interval* and the maximum number of jobs to be managed at the same time.
2. The maximum number of jobs is selected from the pool to be submitted and managed through the DRMAA API. For this purpose, the *producer* method is called for every element selected. The resultant dictionary describing the job is translated to DRMAA and subsequently monitored until their termination. However, there is no function in the DRMAA 1.0 specification that returns all the job states in a unique call. Then, the completion of these jobs is sequentially checked in the *polling interval* set by user.
3. Completed jobs are continuously replaced by new ones, and failed jobs are reintroduced into the pool. The verification of every completed job is done by applying its corresponding *consumer* method.

Nonetheless, other program arguments allow user to benefit from GridWay-specific improvements. For example, the configuration of the *suspension timeout* at the remote batch system when grid resources are used. Additionally, requirements and ranking can be added to the jobs without changing the ones set by the developer. However, these last statements are set for all the jobs managed. Thus, the Workload Division and Job Building performed can be considered as *static* and based on FCFS, but it is valuable to obtain the performance obtained by RBs in Job Scheduling anyway.

4.4. Example: calculating NC transport coefficients

Neoclassical (NC) transport calculations are necessary for the complete simulation cycle of the behaviour of plasmas inside both tokamaks and stellarators facilities. This section presents the porting process to the grid and the computational optimisation of the Drift Kinetic Equation solver code [244] (DKES), devoted to obtain the monoenergetic diffusion coefficients of NC transport, as well as the design and implementation of the DKEsG framework devoted to automatically manage the workflows needed to estimate the final transport coefficients. The advantage of this framework over the Monte Carlo approaches also ported in this work, such as ISDEP or FastDEP, is that

DKEsG allows the estimation of all the transport coefficients, not only the diagonal ones. However, the management of calculation is more complex due to the number of parameters and their possible combinations. Physics related to transport, as well as the physical results obtained in this work are compiled in Appendix C. In this sense, to understand the implemented workflow, it is only needed to know that the final objective of the calculation is to obtain a set of accurate NC transport coefficients, L_{ij}, given by:

$$L_{ij}\left(n,T,\phi\right) = -4{,}552878\left[T_i\right]^{\frac{3}{2}}\left[M_i\right]^{\frac{1}{2}}\left(Z_i\frac{dp}{dr}\right)^{-2} \times \left[\frac{2}{\sqrt{\pi}}\int_0^{\infty} K^{i+j}e^{-K}\hat{D}_{ij}dK\right]$$

which can be used to estimate the physical fluxes. Moreover, the monoenergetic diffusion coefficients $\hat{D}_{ij}$ are obtained from DKES, which requires two parameters, as is stated in [244] and [245]:

$$\hat{D}_{ij} = \hat{D}_{ij}\left[\frac{\nu(K)}{v_T\sqrt{K}}, \frac{E_s}{v_T\sqrt{K}}\right] = \hat{D}_{ij}\left[CMUL, EFIELD\right] \tag{4.1}$$

Therefore, the calculation is structured as a workflow, which is composed of two computationally intensive applications. The first one is the code derived from DKES, (renamed as DKEsG-Mono, and independent of T and n), used to estimate the monoenergetic coefficients, $\hat{D}_{ij}$. The second one is the program that calculates the final NC transport coefficients, L_{ij}, that had to be implemented from scratch and is called as DKEsG-Neo.

4.4.1. Adapting DKES code to run on grid resources

The process described in this section illustrates the usual steps that a developer perform to adapt codes to run on grid infrastructures, which resources usually have a concrete configuration, but lacks required software. Therefore, developers should try to generate application binaries with as less dependences as possible. However, this effort cannot be needed on cloud if tools to set up a customised virtual environment are available.

Original code

The variational version of DKES [244] is the code previously available at the Spanish National Fusion Laboratory, where TJ-II is allocated. This application was originally optimised to be run on HPC architectures already in use on the date when the software was written in Fortran 77. These were CRAY-1 and CRAY X-MP since the code date is 1989; similar information can be deduced for the libraries, e.g. LINPACK [246]. Later, due to the natural evolution of the computational architectures, these systems were replaced by more powerful machines, so the code was adapted to run on an SGI Origin 3800, where it has been executed to simulate the kinetic transport of several fusion devices until now. Moreover, because of its implementation for old fashioned platforms, it is difficult to manage the application itself; thus, a manual reconfiguration and recompilation of the main code is continuously necessary in order to prepare every simulation.

However, the parametric and sequential nature of DKES theoretically allows the division of the problem into independent subtasks, i.e. each parameter calculation can

Table 4.2: Memory consumption and time spent by one task of DKEsG-Mono sorted by number of Fourier/Legendre modes and radius (with a coupled order of 5).

TJ-II normalised toroidal flux ρ (0-1)	TJ-II index for radius (cm)	Min. Num. Fourier modes	Num. of Legendre modes	Memory Consumption (KB)	Xeon 5160 3 GHz (Launch date: 2006)
0.021	4	343	100	13,140	39.704 s
0.021	4	343	200	14,088	1 m 17.528 s
0.021	4	343	1000	21,684	6 m 22.112 s
0.050	8	470	100	23,600	1 m 41.408 s
0.079	12	708	100	51,684	7 m 38.699 s
0.107	16	1252	100	157,420	70 m 6.408 s
0.118	17.5	1751	100	305,232	192 m 51.545 s

be performed by one job that can be run on a grid environment. As an advantage, DKES returns only normalised diffusion monoenergetic coefficients $\hat{D}_{ij}$, which are independent of plasma temperature and density (T and n). So the results can be stored in a unique file or database, indexed by the parameters described in Equation 4.1, which are only functions of the considered magnetic surface. In this way, results can be calculated once and re-utilised for many variations of K, ν and E_s, saving computing time in future simulations.

New x86 binary

In order to be able to execute DKES in most of current computing architectures as possible, several modifications have been made. Thus, the first step has been to port the code to the x86 Linux platforms, since this architecture is nowadays the predominant in scientific environments and particularly in grid infrastructures such as EGI. So, optimisation procedures implemented for old architectures and calls to obsolete packages have been deleted and replaced by similar ones, adapted to the new computational environment. For example, SCSL libraries for IRIX have been replaced by the open source LAPACK/BLAS [247]. After this, as many libraries and software optimisation tools related to the processors as possible have been included in the new executable, in order to efficiently perform the calculations on the grid. This is so for avoiding further incompatibilities and for reducing the number of versions that must be produced, because even when a larger package to be submitted would be created, its size is negligible in this case for the current wide-area networks.

As a result, it has been obtained better performance of the new software on x86 with a combination of compilation options and compilers. GCC 3.4 was used to compile the 3.1.1 version of LAPACK/BLAS libraries and Intel Fortran 9.1 was used to create a final executable file (1.3 MB), the so-called DKEsG-Mono module. The binary can be efficiently run on any Linux distribution from the kernel version 2.4 and on any PIII processor or higher, 64 bits and AMD included. Technically, this module can be executed with a maximum of 6,500 Legendre polynomials and 4,000 Fourier modes. Its execution needs an input file of $\sim$ 5 KB and produces output files of $\sim$ 10 KB.

Verifying feasibility

Nevertheless, the memory size allocated by DKES for one single task depends on the number of Fourier modes used to describe the magnetic configuration of the

reactor. These data increase with the length of the considered plasma radius. However, the size of variable arrays can not be dynamically allocated in Fortran 77, and this was the main reason for the necessity to recompile the code for every simulation.

Porting DKES to Fortran 90 avoids continuous recompilations, nevertheless the amount of memory required by some DKES calculations could become a serious limitation. This is because grid sites have heterogeneous configurations in their worker nodes, with different memory sizes, and there might be not enough resources that meet the DKES requirements. For this reason, the memory consumption by a DKES task has been empirically measured for a minimum number of Fourier modes per radius in a reactor. Data related to typical radii in TJ-II are show in Table 4.2 and consequently the suitability of common DKES calculations for their distribution can be guaranteed on current production infrastructures. In particular, this affirmation is obvious in those resources supporting the LHC experiments, since their processing software requires at least 2GB per core and their virtual organizations are the most widely extended in EGI.

It is noteworthy to mention that since 2012 [240] the DKEsG-Mono module is actually derived from the most updated DKES code [245, 248]. Although this release can run on x86, this feature only simplifies the process of code adaptation, which had been performed again, resulting into an executable, inputs and outputs with similar size, but achieving better performance while returning more accurate results (see Table 10.2 in Chapter 10).

4.4.2. Implementing the transport coefficient calculation

The three final values of L_{ij} were then obtained by integrating the new diffusion coefficient function inside Equation 4.1. The accuracy of the result will depend on how well the analytical function is fitted from the experimental data as well as on the chosen method of the quadrature subroutine; the latter is based on Romberg integration, which uses the generalised open formula of the midpoint rule to do the numerical integration, together with the extrapolation of the function.

The newly implemented module has been called DKEsG-Neo since it returns L_{ij}, the NC coefficients. Two types of meaningful results can be obtained by executing this module. The lightweight one is to force it to calculate the NC transport coefficients for different values of the electric field. For this purpose, less than 10^3 monoenergetic coefficients (D_{ij}) are usually needed, and DKEsG-Neo lasts 3 minutes to estimate each L_{ij}.

When the study requires obtaining the NC coefficients that allow calculating the final NC fluxes and currents of a fusion device, the calculation of every L_{ij} requires a large number of monoenergetic coefficients. This number is typically larger than 10^4 to correctly fit the integral and solve it. The time for calculating each L_{ij} can be over 20 minutes in a machine with the characteristics mentioned in Table 4.2.

4.4.3. Combining jobs into a single workflow

The normalised collisionality ($CMUL = \nu(K)/v_T\sqrt{K}$) and electrical field ($EFIELD = E_s/v_T\sqrt{K}$) pair represents a single calculation for DKEsG-Mono. The parameter sweep variation will be based on calculating a high number of combinations of them within two pre-established intervals. Moreover, DKEsG-Mono returns different results per combination of magnitudes such as radius (r, normalised in ρ), Fourier and Le-

gendre polynomials and other parameters, so the amount of input and output files can grow dramatically up to 10^{12}.

Thus, it was necessary to design a methodology for dividing the DKEsG-Mono calculations and recompose all the collected output files in a coherent way to allow the subsequently selection of the data input for the DKEsG-Neo module. For doing so, auxiliary programs for the automatic generation of indexed inputs, output compiling, recycling results and plotting were implemented in the DKEsG framework.

Furthermore, the computationally intensive parts of DKEsG must be completely executed on the maximum number of grid resources to reduce the final simulation time. As commented above, the calculation is performed by DKEsG framework in two stages. The first one consists of a collection of jobs from which some of their output values ($\hat{D}_{ij}$) are taken as inputs by other jobs for calculating a specific L_{ij} in a second stage. This means that the execution of the different DKEsG modules can be considered as a directed acyclic graph (DAG) workflow. The complete flow chart of the DKEsG framework is shown in Figure 4.1. The magnetic configuration of the reactor is represented by different values of $\overrightarrow{B}$ per radius r, which should be provided as input data by the user or by an external application as VMEC [249]. For each $\overrightarrow{B_r}$, a wide set of values (l_{max}) of $\hat{D}_{ij}$ is obtained by running several $l_{max}/(m-n+1)$ DKEsG-Mono module instances with different values of ν (collisionality) and E_s (electric field). As K_l is the concrete energy that determines the $CMUL$ and $EFIELD$ pair, the $[l_n \cdots l_m]$ is the chunk interval to build the BoT to be submitted to the grid within one DKEsG-Mono job to obtain the $\hat{D}_{ij}$ coefficients. Thus, $m-n+1$ is the maximum size of the BoT set by the user launching DKEsG.

The previously calculated and stored $\hat{D}_{ij}$ coefficients are used by DKEsG-Neo module for solving the L_{ij} integration. T,n, and ϕ could be directly provided or obtained from other applications, devoted to heating plasma simulation, such as MaRaTra [250] or FAFNER2 [67, 224, 225, 226] codes, that can also run on grid as well as VMEC. Other sources can be used, for example, plasma transport evolution codes such as ASTRA [251], but it has to be executed locally. In order to use the outputs from other fusion applications, they must be previously adapted to the DKEsG format by a parser tool.

It is noteworthy to mention again that a full description of the whole workflow and the related physics can be found in Appendix C.

4.4.4. Managing jobs

The implementation of this workflow is not the final step in the porting process to grid. To achieve this goal it is necessary to choose the most efficient and straightforward way to run DKEsG modules on grid and retrieve its results as long as their inter-dependences keep preserved.

GridWay CLI and templates

Besides of its demonstrated performance [135] and compatibility, GridWay performs a reliable and unattended execution of jobs, which it also includes handling DAG based workflows [161]. Moreover, GridWay provides the programmer with some functionality to trigger the migration of a job when its performance falls under a specified threshold. For these reasons, the first release of DKEsG [230, 234] was based on the generation of GridWay templates.

Figure 4.1: Formal workflow scheme for the DKEsG framework, where the functions $r_1 = 1$ and r_{max} represent the order from the minor radius a considered. $\overrightarrow{B_r}$ is the magnetic configuration for a radius r provided by means of Fourier series. l_{max} is the number of DKE calculations performed for a specific $\overrightarrow{B_r}$, and $(m - n + 1)$ are the chunk interval of these calculations submitted to the grid within one job. $\hat{D}_{ij}$ represent the normalised diffusion coefficients for a concrete energy K_l and magnetic configuration and radius $\overrightarrow{B_r}$. T_p is the temperature and n_q the density in a certain plasma state q and p. The potential is represented by ϕ. See the text and Appendix C for further explanations.

Therefore, two specific programs have been implemented to manage the DKEsG execution on a grid environment, beside the one managing the workflow. The former is executed on the user interface (GridWay machine), it performs the Job Building and it checks the exit code and the output files retrieved by each DKEsG module done on the grid. If any of them is not correct for a DKEsG-Mono task, the program resubmits not only the failed job, but all DKEsG-Neo jobs that depend on it. Note that GridWay maintains the workflow dependences when a job is migrated but not when a job fails. In this sense, the time spent by a module running on a grid site is also measured to satisfy a certain fixed threshold, if it exceeds this limit, the job is forced to migrate.

The second program is a simple monitor that is executed jointly with either DKES-Mono or DKES-Neo on the worker node at the remote grid site. It is in charge of controlling the walltime, checking the generation of intermediate output files and triggering a self-migration for performance reasons. It works independently to the first program, being more able to quickly detect performance slowdowns than failures.

DRMAA

Nevertheless, the experience with long computations demonstrates that the mechanism based on the generation of templates is not sustainable over time. The wide range of extra parameters of these modules and their combinations made the workflow too complex to manage the inputs, the results and the precedence constraints among DKEsG-Mono and -Neo jobs. Their volume requires storing them in a database that must be exhaustively controlled. By means of this database, DKEsG can detect the completion of a parameter scan for a certain plasma surface to trigger the submission of the corresponding DKEsG-Neo job.

Therefore, the final release of DKEsG uses the standardised producer-consumer library presented in Section 4.3. For this purpose, two *application* classes were implemented, one for DKES-Mono and the other for DKEsG-Neo. The *input pool* only guides the workflow, being the *producers* and *consumers* of the *applications* in charge of getting the inputs from, and storing the results into the database. The remainder mechanisms to parse inputs or manage files are maintained and included in their respective methods. This version is used in the experiments described in Chapters 9 and 10.

4.5. Executions on grid with DKEsG

The objective of performing these experiments is two-fold. The first one is to evaluate the suitability of the adaptation of DKEsG to distributed environments, being able to obtain meaningful physical results. Moreover, the main intention is to evaluate the overheads introduced, the reliability obtained, the IS trustworthiness and how the RB (in this case GridWay) deals with the unexpected issues.

As explained through the last Section and formally in Appendix C, the DKEsG-Neo module needs the outputs of many DKEsG-Mono executions to calculate the NC transport coefficients (L_{ij}). Additionally, the outputs ($\hat{D}_{ij}$) from DKEsG-Mono can be stored for further studies and have their own physical importance and subsequent discussion. For these reasons, the DKEsG framework has two operation modes: standalone DKEsG-Mono executions and the workflow mode. Therefore, these options will allow the study of the performance when parameter sweep and workflows are

used relying on a RB such as GridWay.

4.5.1. Test bed and common parameters

The EELA-2 production grid facility, based on gLite middleware, was used for performing the tests explained below. Resources belonging to this infrastructure as those when this experiment was carried out are summarised in Table 4.3. It counted on 12 grid sites supporting the *prod.vo.eu-eela.eu* VO. On the other hand, a dedicated machine with GridWay 5.4 was deployed to perform the experiments.

The Heliac stellarator TJ-II standard magnetic configuration was selected for the real calculation. Not only real values for the geometry of the TJ-II (represented with their Fourier coefficients and radius of the magnetic surface) have been chosen, but also some other parameters. Thus, a test for average plasma minor radius $r = 4$ cm has been performed. Other magnitudes such as the temperature, the atomic mass and the potential have all been set to the unity, i.e. T, n, ϕ variables in equation (8). Finally, a set of 100 Legendre polynomials and 343 Fourier harmonics has been used to represent the distribution function and the TJ-II configuration, respectively. In this configuration each DKES execution only uses 14 MB of memory and lasts ~ 40 s following the data compiled in Table 4.2. Consequently, some grouping into BoTs should be performed in both experiments to limit the influence of overheads. Furthermore, these experiments will evaluate the user's capacity to limit these overheads by statically fitting their applications. For this purpose, the first experiment computed very sort jobs and in the last one the BoT size was subsequently fitted. Additionally, the maximum number of submitted jobs per provider (the $MAX\_RUNNING\_RESOURCE$ option in GridWay configuration) was reduced in the second experiment. The intention is to show that more resources do not have to imply better speedups.

Other default values in GridWay configuration are maintained, with exception to the number of jobs allowed to be scheduled at once, which was unlimited. With this last option, the study of the Job Scheduling field can be performed without care of the overheads related to the application, because GridWay has to deal with all the jobs since the beginning of the experiments.

4.5.2. Parameter sweep calculations

The collisionality and electrical field pair represent a single calculus for DKEsG-Mono. The parameter sweep variation will be based on calculating a high number of collisionality values within an interval per electrical field. In this experiment, a variation of 500 collisionalities per 5 electrical fields has been calculated. Thus, a total number of 2,500 DKEsG-Mono executions has been performed, the physical results of which are shown in Figure C.1 and explained in Appendix C.

For the sake of comparison with the grid execution, this calculation has been performed by a unique DKEsG-Mono job using a single core of an Intel Xeon 5160 (with 3.00 GHz and 4 MB cache) such as the one used for building Table 4.2; on a free machine it lasted for 26 h 7 m 35 s. This type of machine is the same as the resources offered by the *ce-eela.ciemat.es* (see Table 4.3), and one of the most powerful offered by the whole providers at the moment when our grid test was performed. Regarding these issues, the adequate chunk size of DKEsG-Mono calculus was estimated to be of 5 tasks, resulting into 500 jobs. This value was obtained considering that five collisionality-field pairs ($<$ 4 minutes in aforementioned machine) is the appropriate

Table 4.3: DKEsG-Mono executions on every resource available on the EELA-2 infrastructure.

Site	Status	Total	OK	F	Err	P&S
axon-g01.ieeta.pt	OK	43	25	18	13	5
ce01.eela.if.ufrj.br	OK	121	118	3	1	2
ce01.macc.unican.es	up/saturated	44	4	40	1	39
ce.eela.cesga.es	OK	78	16	62	1	61
ce-eela.ceta-ciemat.es	up/saturated	80	0	80	0	80
ce-eela.ciemat.es	OK	269	248	21	0	21
ce.labmc.inf.utfsm.cl	OK	78	62	16	0	16
grid001.cecalc.ula.ve	maintenance	48	0	48	48	0
grid012.ct.infn.it	up/saturated	74	0	74	0	74
gridgate.cs.tcd.ie	up/saturated	55	5	50	22	28
kuragua.uniandes.edu.co	OK	59	22	37	0	37
ramses.dsic.upv.es	not matched	Under the 1.4 GHz threshold				

Total means the total number of submitted jobs, OK the successfully ended, F stands for failed, Err for error and P&S for Performance and Suspension (deeper explanation through the text).

number to evaluate the feasibility of the distribution of our DKES version among providers, as well as to compile information about the operation with the RB. Furthermore, the appropriation of resources was only restricted to 40 slots per site and GridWay was allowed managing the 500 jobs at the same time. The intention of these aggressive criteria is to measure the performance obtained for the application, flooding the infrastructure with very short jobs, which make use of the maximum number of available resources.

The performance thresholds of DKEsG monitor programs have been set as follows: the maximum real wall-time of a DKEsG-Mono job into a worker node to 25 minutes; the maximum time spent at grid sites to 30 minutes; the suspension timeout for dispatching the job by the remote LRMS to 5 minutes; and, the minimum usage percentage of the CPU (core) assigned to a DKEsG-Mono job to 80 %. Moreover, the resource priority for every job was set relying on the CPU speed of providers following the information given by the IS of the infrastructure, but ruling out resources lower than 1.4 GHz. Therefore, this configuration of the application allows lower powered resources to compute the jobs 7.5 times slower than the reference machine. Obviously, none machine under the 1.4 GHz lasts so long in compute one job if the resource are exclusively assigned. Thus, the monitors will detect inconsistencies of the IS or overbooking of providers.

Table 4.3 shows where a DKEsG-Mono job was successfully executed (OK) and where the a job was failed (F). It is important to bear in mind that these failed jobs do not necessary indicate a misconfiguration at remote grid sites. These sites could be temporary saturated of other jobs belonging to other users; or they could fail because either an excessive suspension time was produced, or a lower CPU power than the expected by the DKES monitor was observed. Then, due to the strict thresholds fixed for this test, an increasing number of migrated and failed jobs is experimented. To easily differentiate these jobs from the ones originated by other reasons (network cuts, hanged worker nodes, misconfiguration...), such a kind of failures are declared separately in Table 4.3 as "P&S" (performance and suspension) and "Err" respectively.

Time spent by the complete test on grid was 52 m 47 s. Therefore, it supposes getting the final compiled output of DKES 29.7 times faster than its sequential version running isolated on the most powerful worker node of the infrastructure.

Besides the impressive reduction on the processing time obtained in the perfor-

med test, a key factor is important to be taken into account to really evaluate the final improvement, i.e. the time lost by grid middleware operations. While the real accumulated time in the test for DKEsG-Mono calculations was 45 h 6 m 33 s, the accumulated time spent by successfully ended DKEsG-Mono task was 72 h 50 m 25 s. This represents an overhead of 61.48 % respect the useful execution time.

Thus, to maximise the performance of the whole calculation, this previous percentage can be reduced by increasing the number of calculi by DKEsG-Mono job. This reasoning can be easily deduced observing Figure 4.2, where the intrinsic overhead (waiting queues and middleware) varies for the different sites due to their own configuration and system load, but can become negligible when real execution time rise to a certain level. The increase of the BoT size is discussed in the following Subsection 4.5.3.

Despite of this, the scheduling configuration of GridWay (this is, the weight policy assigned to the more used resources or from where the resources have been excessively migrated) has effect on the accumulated time wasted in waiting for, detecting and migrating failed jobs, which is compiled in 91 h 31 m 5 s. Although it can be a specific fitting for the certain employed infrastructure, it will not change through the time. Even more, if GridWay were shared among users, this fitting is performed by administrators. Subsequently, it will be completely independent from the application to be submitted because users cannot modify the scheduling configuration. For this reason, fitting Job Scheduling for individual calculations is not feasible with RBs such as WMS and Condor-G.

Moreover, P&S failures indicate that the real availability offered by providers are not the one finally supplied. In Figure 4.2 only the resources where the production has been more successfully executed are shown (the ones with the *OK* status in Table 4.3). The impact of failures on these sites is indicated as "fail overhead". The variability of the measurements of the weight of P&S on those failures implies two conclusions: IS are not trustworthy and the overheads depends on the current status of every provider, which only can be known with continuous tests.

4.5.3. Running in workflow mode

In this experiment, a variation of 500 collisionalities per 9 electric fields has been performed. Therefore, a total number of 4,500 independent estimations of the $\hat{D}_{ij}$ coefficients have been carried out by the DKEsG-Mono module. The obtained normalised monoenergetic diffusion coefficients were utilised by the DKEsG-Neo module for calculating the three main L_{ij} per electric field.

Regarding the issues mentioned in previous Subsection, the chunk of the DKEsG-Mono calculations was set to 45; this value was obtained considering that these 45 collisionality-field pairs ($\sim$ 29 minutes and 47 seconds in the machine type mentioned in Table 4.2) can be the suitable lower execution time limit to evaluate if the speedup is improved by only decreasing intrinsic overheads in a typical simulation on the grid.

Therefore, the whole DKES-Mono calculation is contained in 100 jobs. On the other hand, the three DKEsG-Neo jobs that calculates the three L_{ij} associated to one electrical field spend $\sim$ 3 minutes each, and they sum up 27 jobs. They are automatically submitted to providers by GridWay when its dependences have been successfully achieved. Note that the DKEsG release based on the GridWay CLI is used in this experiment.

Moreover, the performance thresholds on the monitoring programs have been fitted

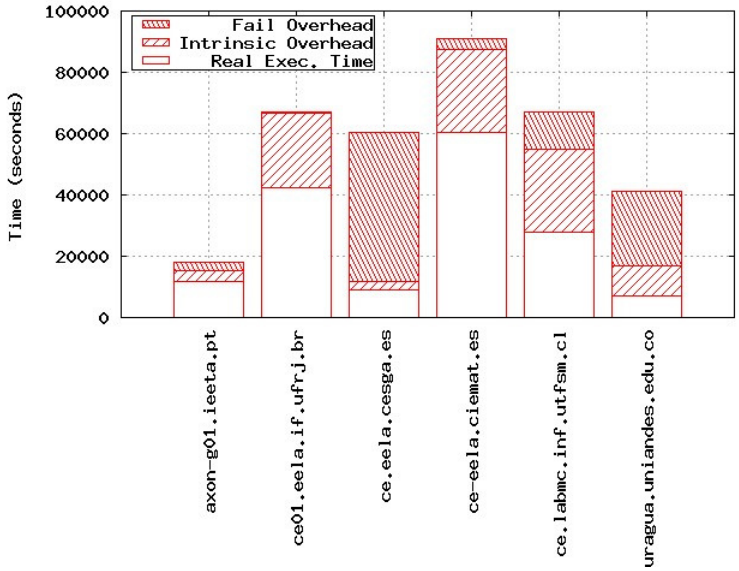


Figure 4.2: Real execution time of DKES compared with the associated middleware overhead (intrinsic) of the successfully executed jobs, including staging-in and stage-out files. Also the time spent by failed jobs is depicted (deeper explanation through the text).

for this new BoT size: the maximum real wall-time in a working node has been limited to 90 minutes for DKEsG-Mono jobs and 5 minutes for DKEsG-Neo jobs; the dispatching suspension timeout at the remote batch systems has been limited to 5 minutes in both cases; and, the maximum time spent at grid sites to 95 and 10 minutes respectively. Therefore, the allowed execution time was reduced for lower powered processors, being 3 times more than the one expected by the reference machine, but the maximum waiting time in queues is maintained.

On the other hand, the ranking relying on the CPU speed published and the discarding of those resources bellow 1.4 GHz have been maintained for this experiment. However, the maximum resource allocation has been restricted to 10 slots per site. Thus, the maximum number of nodes that GridWay can allocate is 4 times fewer than the previous experiment.

Time spent by the complete test on grid was 2 h 51 m 53 s. This result demonstrates a true distribution of jobs since 10 grid sites in the EELA-2 infrastructure have processed more than 5 jobs each. For the sake of comparison with the grid execution, if this test were performed by a unique DKEsG-Mono job and its results processed sequentially by the DKEsG-Neo program using a single core of a free machine similar to the one described in Table 4.2, it would have lasted for more than two days and one hour. Therefore, it supposes getting the final 27 L_{ij} coefficients 17 times faster than its sequential version running isolated.

However, this speedup cannot be directly justified when the maximum number of resources had not been reduced at the same degree. Moreover, the 27 DKEsG-Neo jobs increase the makespan because they must wait for the completion of many other jobs to be submitted. The suspension timeout (the main weight in intrinsic overhead) is maintained, thus the intrinsic overhead of successful jobs are reduced to the $\sim$ 10 %. However, there have been many failures and this type of overhead depends on

the real availability of the providers at any time, and it is clear that GridWay cannot correctly handle this issue with the data shown by IS, neither by the information compiled in its accounting databases. It is needed more advanced tools that perform the characterisation of providers with respect to the application, this is, benchmark sites, profile executions and foreseen availability.

It is noteworthy to mention the physical results obtained from the DKEsG-Mono module and the NC transport coefficients calculated by DKEsG-Neo module using them are shown in Figures C.2 and C.3 in Appendix C.

4.6. Conclusions

The main objective of this chapter is to evaluate the available mechanisms and tools for adapting an application to highly distributed environments. The conclusions can be summarised in:

- The use of CLI and even standardised APIs are embarrassingly for developers; skeleton approaches are useful to overcome this issue.
- WfMs can be useful to ease the adaptation of certain complex applications, but do not allow users to modify the underlying Job Scheduling.
- RBs are an excellent entry point to profit from DCIs, achieving important reductions on the final processing time of distributed applications with respect to the serial ones, but they do not properly manage the associated overheads because they only work on Job Scheduling and lack infrastructure characterisation.

Therefore, other tools that manage whole Workload Scheduling has to be evaluated for early-binding. Optimising the overhead percentage with respect to the real execution time is in the scope of self-schedulers, an aspect that will be properly evaluated in Chapter 6.

All these conclusions have risen as a result of a main achievement performed in this chapter: the adaptation of several applications to highly distributed environments, which will allow communities to face new challenges and uses. In particular, most of applications have been adapted relying on the standardised DRMAA API, being able to be run on LRMS or grid (and potentially on cloud, as will be explained in next Chapter 5). In this sense, a distributed version derived from the DKES [244] code, the DKEsG framework, has been successfully run on grid. This step opens new possibilities for fusion community since, from now on, the NC transport coefficients can also be calculated on the grid reducing the time consumption and increasing the feasibility of the code use. This is a key factor for those researchers who do not have an easy access to huge computational centres.

Chapter 5

Scheduling Straightforward Executions in Clouds

5.1. Introduction

GridWay [10] is a meta-scheduler so flexible and "agnostic" enough to potentially manage any kind of resource if it is extended with the necessary modules. Thus for a cloud federation, GridWay could use the information that dynamically updates the IS to select the most suitable cloud provider every time. For this purpose, the Scheduler should take account of the requirements set in the description of common jobs, which execution inside a controlled virtual environment, i.e. a VM, actually is the final objective. Therefore, the approach is to create a VM per every job to be executed, as well as grid sites enables an ephemeral *chrooted* environment to execute these jobs.

Perhaps the earliest approach that schedule jobs among remote resources as they were cloud providers can be found in [233]. In this paper, a mechanism that makes use of the existent grid middleware to schedule VMs that contain some software in remote sites is proposed. GridWay takes account of the state and characteristics of sites to submit a wrapper encapsulated into a regular grid job that is able to boot a virtual machine. OS images were directly uploaded to sites or to storage elements through protocols such as GridFTP. Obviously, this approach does not exploit all the advantages that virtualisation offers, and only was justified by the absence of cloud interfaces and middleware.

The standardisation process that makes possible the establishment of cloud federations also opens the door to GridWay to directly make use of cloud resources. In consequence, the advanced scheduling features of GridWay, its usability and compatibility will really come into cloud. For this purpose it is necessary to implement two new information and execution drivers able to manage cloud providers. The solution [242, 252] differentiates from other approaches based on GridWay cited through the related work [233, 42, 43, 30] in its Scheduler, which takes the decisions of where VMs are started as well as GridWay manages the VMs with cloud middleware. Benefits include:

a) Automatic discovering and monitoring of providers that belong to several cloud federations.

b) Scheduling capabilities based on constraints, ranking, fair-sharing, deadlines, etc., to instantiate VMs at providers with certain characteristics, like:

   - specific VM image (e.g. by the *appdb.egi.eu* identifier);
   - available hardware, with advanced reservations;
   - associated costs and budgets, QoS, etc.

c) Grid security standards and protocols are preserved to enable compatibility with external grid services.

d) Direct execution of jobs.

e) Minimal contextualisation, fully customisable by the user.

f) Compatibility with legacy applications.

However, time spent in every VM instantiation can represent an excessive added overhead comparable to the waiting times in LRMS queues at grid sites. For this reason, the suitability of the approach for short jobs is experimentally studied in Section 5.4.

5.2. Early developments

The scheduling, deployment and execution management of single VMs are possible in a grid infrastructure using the customary middleware installed in grid sites. The approach presented in this section consists in encapsulating a virtual workspace in a grid job managed by GridWay. It also incorporates the functionality offered by a general purpose meta-scheduling system. So, the genuine characteristics of a grid infrastructure (i.e. dynamism, high fault rate, heterogeneity) are naturally considered in the proposed solution.

The description of the grid job must include all the requirements of the virtual machine within. So, the job template used by GridWay must specify, for example, the Xen Hypervisor version (that must be compatible with the Linux kernel needed by the virtual machines), and other hardware requirements (e.g. free memory or CPU load).

GridWay schedules a grid job as follows: the *Information Manager* (see Figure 2.3) holds a list with the available resources in the grid and their characteristics. This list is periodically updated by querying the information services to monitor and discover grid hosts. The *Dispatch Manager* filters out those resources that do not offer the needed hypervisor, or do not have free slots or do not meet any other job requirement. Then it sorts the remaining hosts according to a user-supplied rank expression. The highest ranked resource is used to dispatch the job.

5.2.1. Deployment of virtual machines

In general, the images could be downloaded, as any other input file, in the *prolog* phase from the client or from an image repository, using GridFTP or any other grid protocol. Additionally, GridWay allows the definition of an optional *pre-wrapper* phase to perform advanced job configuration routines. This process consists in an

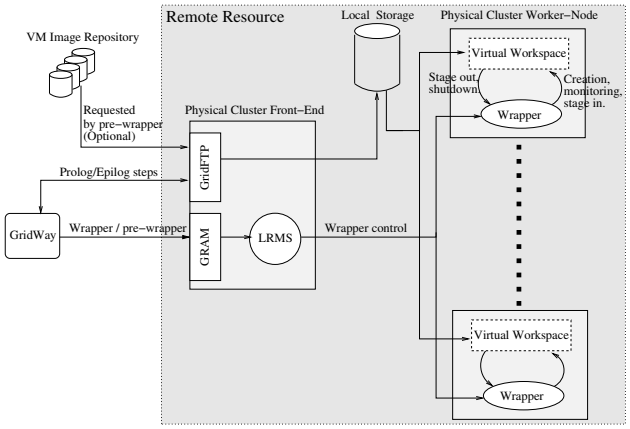


Figure 5.1: Schematic representation of the job execution process within virtual machines deployed in a grid site.

user defined program that is executed on the cluster front-end (the gatekeeper of grid site). In our case, this program checks the availability of a given image, and transfers it from a GridFTP repository if needed. Note also that higher level data services, like the Replica Location Service (RLS), could be used.

Then, the *Execution Manager* interfaces the gatekeeper service (through GRAM2/ 4/5, CREAM or BES) to submit the *wrapper* program, which performs the following actions:

1. It checks the availability of the requested VM image in the cluster node.
2. The VM is started or restored with an unique identification and MAC address. Then, the *wrapper* waits for the VM activation by periodically probing its services.
3. The *wrapper* copies all the input files needed by the experiment to the VM, and it executes the scientific application.
4. The output files are transferred back to the physical cluster file system to be copied to the client host in the *epilog* phase.
5. Finally it shuts down (or suspends to disk) the VM.

The previous process is depicted in Figure 5.1.

5.2.2. Multiple weaknesses

This strategy does not require additional middleware to be deployed with exception to the virtualisation hypervisor, as it is based on well-tested procedures and standard services. Moreover, it is not tied to a given virtualisation technology. However, it presents important drawbacks:

- The underlying LRMS and middleware are not aware of the nature of the job itself. Therefore, some of the potential benefits offered by the virtualisation technology (e.g. server consolidation) are not fully exploited.

- Administrators of remote grid sites have to allow grid users to gain enough super-user privileges to boot VMs.
- VMs should be pre-configured before their instantiation, i.e. they are not benefited from the contextualisation mechanisms.
- IS can include specific tags to advertise that the virtualisation capacity is allowed, but they will not describe grid sites with the characterisation suitable for cloud providers.

In any case, the presented design can be used to illustrate the questions derived from following one-job per VM approach, as will be performed in Subsection 5.4.1.

5.3. The current approach on IaaS clouds

With the advent of cloud federations and their related protocols and mechanisms, using the aforementioned techniques makes no sense. However, the previous approach is not completely outdated, because the achievement in allowing VM scheduling among providers can be applied on current cloud infrastructures. For this purpose, two drivers have been developed to interface with cloud services. The first one is devoted to filter and compile the specific data related to cloud characteristics from the IS. The other manages the VM creation and the job execution. Both are fully described through the following subsections.

5.3.1. The GWcloud Information Driver (ID)

This new driver looks up for cloud providers in top BDIIs of one or multiple federations. The user can configure the search to constraint the matches to certain characteristics published by providers. Currently, the driver supports the EGI FedCloud, but it can be modified to directly use OCCI or AWS interfaces to work on a multi-cloud environment. Subsequently, the driver filters the information to dynamically notify GridWay about the characteristics of providers in which the user is authorised. Every provider found is included as an independent resource in the Host Pool. Thus, the information can be consulted by the user through the GridWay commands and it is shown as:

- The URI contact endpoint, the protocol, hypervisor and VIM release, the maximum number of available cores, etc.
- Every OS template name (the *os_tpl*) and its *appdb.egi.eu* image identifier are compiled in a list of pairs and included as a new tag.
- Every resource template (*resource_tpl*) is shown as a different queue, with its own characterisation: number of cores, memory, etc.

5.3.2. The GWcloud Execution Driver (ED)

This driver enables the direct execution of a conventional grid job in a VM exclusively instantiated for this purpose. The driver can utilise the user's proxy credentials because it runs in the user-space mode. This allows using resources from federated clouds based on X.509 and VOMS. Additionally, the proxy is contextualised to be

remotely used by jobs to access grid services. To preserve its integrity, the contextualisation file is encrypted, restricting the access only to secure OCCI services. On the other hand, the rOCCI-client [253] is used to perform the operations against the providers. Therefore, when the Scheduler chooses a cloud provider to execute the job, the driver performs the following steps:

1. It gets and stores the match, i.e. the description of the job and the URI of the OCCI service.
2. It interprets the job description to obtain the inputs, outputs and executable URIs, the *os_tpl*, and the *resource_tpl*.
3. Contextualisation: It makes a Cloud-Init file that includes:
   a) creation of a new user with *sudo* privileges;
   b) creation of a file with the temporal user proxy;
   c) inclusion of the EUGridPMA[1] repositories;
   d) pre-installation of certificates from CAs and minimal grid tools (*globus-url-copy*);
   e) shell lines needed to download inputs, execute the job and store the outputs (i.e. through GridFTP or the Globus GASS protocols).
4. It builds and performs an OCCI *create* operation that includes the contextualisation file, the *resource_tpl*, the *os_tpl* and the URI of the provider. Subsequently, the job is considered in a *PENDING* state.
5. It waits for the VM starting to change the job state to *ACTIVE*. To make this periodically, it performs OCCI *describe* operations. If this circumstance does not happen during the timeout set in the job description, the job is considered as *FAILED*.
6. When the VM is running, the driver waits for the VM becoming into *inactive*; subsequently, the job is considered *DONE*. However, if other VM condition is reached, it returns *FAILED*.
7. Finally, it deletes the VM.

Note that a DONE state just only implies that the job was ended. It is the submitter (i.e. the user, some application or the pilot factory) who should interpret the exit status code or the outputs from the job.

5.3.3. Scheduling VMs and jobs

The process described through this subsection finally enables users to submit jobs to GridWay, which are automatically executed in the VMs scheduled for this purpose in the cloud federations. The sequence followed by Scheduler, drivers and providers is shown in Figure 5.2 and summarised as follows:

1. GWcloud ID periodically searches for provider updates at top-BDIIs. When it gets knowledge of changes, it notifies the GridWay Core, which includes the updated data into the Host Pool.

[1] https://www.eugridpma.org

2. The GridWay Scheduler notices that some cloud provider fulfils the requirements of certain job and it is *free*. Then, the Scheduler assigns this job from the Job Pool to that provider and lately, a *SUBMIT* operation is sent to the GWcloud ED.

3. The GWcloud ED stores the matching, interprets the job description and creates the Cloud-Init file. Subsequently, it notifies the GridWay Core (and consequently the Scheduler) that the job is in a *PENDING* state using the *CALLBACK* operation. Then, it performs the *create* operation against the OCCI service of the provider.

4. The VIM running on the provider checks the *os_tpl* availability and the client's permissions and quotas to reserve the virtual workspace described in *resource_tpl*. If everything is correct, it creates the VM. Through the booting process, the Cloud-Init contextualisation is performed and the job starts.

5. The GWcloud ED periodically tests the VM state through a *describe* operation to the OCCI interface of the provider. Thus, it immediately notifies the GridWay Core about any change in the VM state using the *CALLBACK* operation. If the VM crashes, the job becomes *FAILED* for the Scheduler.

6. When the job is *ended*, the VM automatically shuts down and it is detected by the GWcloud ED. The driver considers the job as *DONE*, it notifies the GridWay Core (and Scheduler) and it finally performs the *delete* operation to remove the virtual workspace from the cloud provider.

5.4. Analysing data from the XMM-Newton spacecraft on the cloud

The objective of the experiments performed is to evaluate the suitability of the GWcloud drivers following an early-binding approach. It is of interest to demonstrate that the advantages of creating a personal virtualised environment are preserved. In this sense, applications with excessive dependencies and installation size are difficult to deploy in grid. These applications imply a constant effort by VO managers, who have to continuously re-build the code and install them in every grid site. This problem can be easily solved by distributing virtual images with the latest release of the software. Note also that this will improve the robustness of the software as is always executed and developed on the same environment. However, this approach is not feasible for individual users that make use of their personal applications. Cloud providers usually only offer clean VM templates of certain OS, and they force users to contextualise. The GWcloud approach allows developers to avoid dealing with contextualisation, while they focus their efforts on the configuration and execution of their applications.

For these reasons, the software used for processing the data from the XMM-Newton satellite is selected. A description of XMM-Newton SAS is in Subsection B.3.2 of the Appendix B, but this section put the focus on their computational requirements. Every SAS release increments their installation size, being the current one (14.0.0) of 1.8 GB, and additionally requires external software such as HEAsoft[2](6.17, 202 MB).

[2]http://heasarc.gsfc.nasa.gov/docs/software/lheasoft/

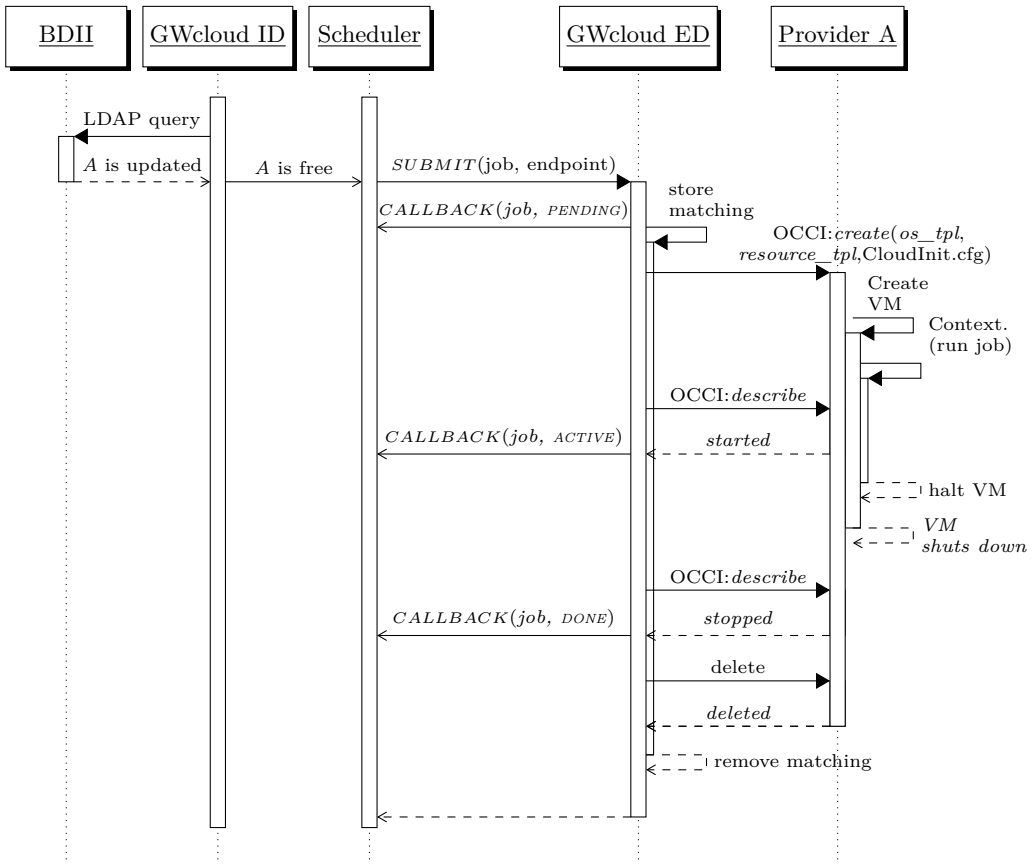


Figure 5.2: Sequence of activities to accomplish any job in a cloud federation with the GWcloud drivers.

Developers of SAS only create compiled versions for few OS releases. On the other hand, besides to the observation data files (ODF, ∼600 MB), SAS needs the current calibration files (CCF, another ∼ 600 MB). Depending on the calculation selected the execution time ranges from few seconds to 30 minutes and the outputs can take more than 500 MB. The sizes provided correspond to the decompressed ones, i.e. when the files are transferred are compressed and they are ∼ 30 % smaller.

Two types of experiments were performed. The first one measures the impact of three virtualisation approaches on the application performance, but on an controlled environment, the analysis of which will be used as an introduction of the costs associated to booting a VM for a single job. The second experiment evaluates the GWcloud approach on an infrastructure in production, i.e. the creation and configuration overheads jointly with the capacity of scheduling VMs across several providers in a federation.

5.4.1. Feasibility of the virtualisation mechanisms

The simplest deployment is to previously store VM images at cloud providers. This approach is followed by many research communities when have a wide implantation or influence on the cloud federation. Thus, although the tests performed are based on the middleware developed in Section 5.2, results are still valid to illustrate the performance loss when:

- virtualising resources;
- virtual clusters are deployed in standalone cloud providers;
- VMs are started or restored to accomplish only a job.

Obviously, the measurements do not show the cost of the contextualisation, post-configuration, virtual cluster elasticity or the scheduling among providers.

5.4.1.1. Test bed description

The behaviour of the previous deployment strategy was analysed on a research test bed based on the Globus Toolkit 4.0.1 and GridWay 4.7. The test bed consists of two resources: a *client host*, and a PBS cluster for processing purposes. The main characteristics of these machines are described in Table 5.1. These hosts are connected by a Fast Ethernet campus network.

The client host runs an instance of the GridWay meta-scheduler and holds the input dataset (∼22 MB) with the ODF and the CCF to be analysed. It also receives the analysis output files (∼13 MB) with the observation events to perform post-processing tasks. The SAS jobs are dispatched to the *front-end* cluster, with two Xen-capable worker nodes (WNs). Note that no virtual machines are started in the cluster *front-end*. Every test consists in performing 100 times the analysis of the same observation from the XMM-Newton satellite. Also, caching of the observation data is avoided.

Implementation details

In the following experiments, the OS images are already stored in the front-end and are accessed in the cluster nodes via NFS. Although this configuration could impose significant overheads, they have not been experienced because of the small size of the cluster used in this work.

The disk images have been obtained from a typical installation with the Fedora Core 4 operating system and the XMM-Newton SAS 6.5 software. In addition, the VM includes a RSH server for executing the remote commands requested by the *wrapper* program. Finally, this disk image is split in three different files to save storage space and ease its deployment (see Table 5.2):

- The root file system, with read-write permissions to modify the virtual machine configuration files at boot time. Two copies of this disk image are available to be simultaneously used by the two cluster worker-nodes.
- The `usr` file system, with the standard Linux applications and libraries. This disk image is read-only and shared by all the cluster nodes.

Table 5.1: Summary of characteristics of the test bed resources in first experiment.

Host	CPU	Memory	OS	Service Configuration
client host	PIV HT 3.2 GHz	512 MB	Fedora Core 4	GT4, GridWay 4.7
front-end	PIV HT 3.2 GHz	512 MB	Debian Etch	GT4, PBS, NIS, NFS, DHCP
WNs	PIV HT 3.2 GHz	2 GB	Debian Etch	Xen3.0 testing

- The `opt` file system, with the XMM-Newton SAS installation. This disk image is also read-only and shared by all the cluster nodes.

Additionally, a local disk image has been created in each WN. The virtual machine mounts this image in the `scratch` directory, where the SAS program stores temporal files and data. So, the input/output operations performed by analysis software are always made in the local hard disk.

The VMs are configured with 512MB of main memory and a single virtual CPU. The network of VMs is configured with a DHCP server, which dynamically assigns private IP addresses different from the ones belonging to the physical cluster.

Table 5.2: Disk layout of the virtual machines in first experiment.

Mount point	Size	Contents
`/`	500 MB	Fedora Core 4 base system
`/usr`	650 MB	Standard Linux applications
`/opt`	600 MB	SAS 6.5.0
`/scratch`	2 GB	Ext3 disk image

5.4.1.2. Results

The overhead induced by the virtualisation following an early-binding approach is studied in this subsection. Additionally, application level metrics have been introduced to analyse the results from the user's point of view.

Execution without virtualisation (Test1)

The first experiment submits the 100 grid jobs through the GridWay meta-scheduler with the standard *wrapper* program, i.e. without virtualisation. In this way, an upper-bound performance is obtained that will be used to study the virtualisation alternatives below. The average execution time of a SAS job with this configuration is 148 seconds.

Persistent virtual machines (Test2)

When a VM per job is instantiated, these VMs cannot be persistent at cloud providers. However, virtual clusters or virtual sites can be deployed in a provider to be accessed following the early-binding approach. Therefore, this experiment measures their associated overheads. Moreover, it shows the inherent cost of virtualisation with the technology available as that of the date when the experiments were performed, which was roughly a 20 % increment in execution time, as can be seen in Figure 5.3.

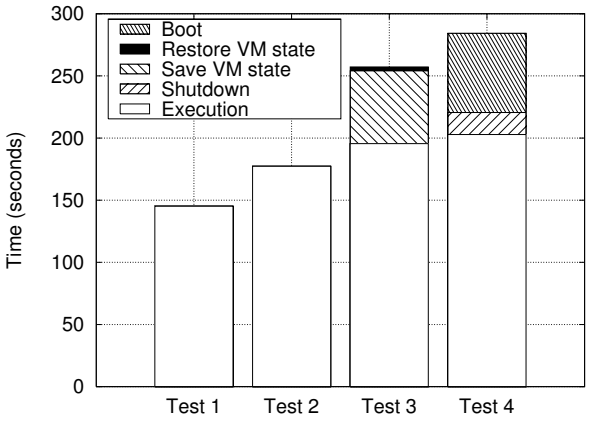


Figure 5.3: Average operation times obtained in the *wrapper* phase: without virtualisation (Test1), with persistent virtual machines (Test2), pausing/restoring (Test3) and stopping/starting the virtual machine (Test4).

Saving and restoring the virtual machine state (Test3)

Unlike Test2, for the modified *wrapper* presented in Subsection 5.2, restoring VMs is similar to booting one VM per job. Moreover, *suspend* VMs are allowed by OCCI specification, thus including this test in the study is valuable for the comparison with the real experiments performed in following Subsection 5.4.2.

The state of a VM includes, in addition to disk images and Xen configuration files, a representation of its main memory and the CPU registers of all its virtual processors. When the context of a VM is saved, it can be later restored, keeping its configuration and resuming the execution of its processes.

This feature, which is efficiently implemented by Xen, can be used to keep the system services in the VM memory, the SAS program, and the related shared libraries. This is done without keeping it active, so system resources are saved. In this case, when the execution of the SAS job ends , the *wrapper* program saves the context of the VM, which is restored before executing another SAS job in that worker-node.

As expected, the execution time is similar to that obtained in the previous experiment (Test2), see Figure 5.3. The additional overhead is mainly due to saving the state of the VM, and implies an overall increment of 77 % in the execution time compared to Test1.

Stopping and starting the virtual machines (Test4)

Stopping the VM after the execution of each SAS job (and starting a new one before executing it) allows a straightforward deployment of the virtual workspaces. However, it adds an additional overhead to the boot/shutdown process of the VMs. In particular, the average execution time is roughly twice that obtained in Test1 (see Figure 5.3).

The problem size of the astronomical observations used in the above experiments has been deliberately chosen small to ease the measurement process. However, in general it will be considerably larger, increasing the total execution time to several hours. As the boot/shutdown and save/restore times are independent of the problem

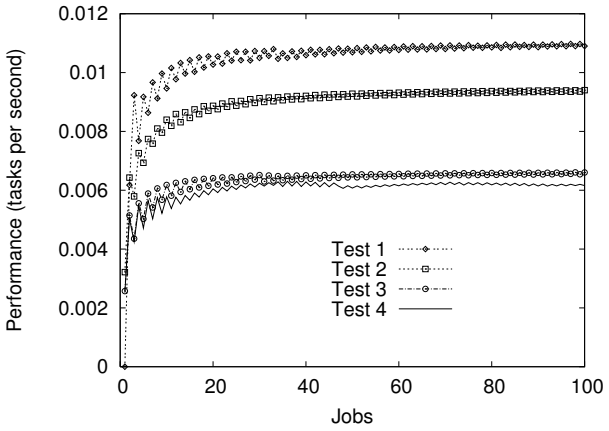


Figure 5.4: Throughput (jobs per second) without virtualisation (Test1), with persistent virtual machines (Test2), saving/restoring (Test3) and stopping/starting the virtual machine (Test4).

size, the additional overhead of Test4 and Test3 will be negligible. Therefore, the overall virtualisation overhead will tend to that obtained in Test2.

It is interesting to analyse the system behaviour from the application point of view. The performance obtained in the execution of a HTC application can be studied using the system throughput $P(n)$, defined as usual:

$$P(n) = \frac{n}{T} \tag{5.1}$$

where T is the execution time of n SAS jobs. Note that the execution time includes both file transfer times (29 seconds on average) and Globus overhead.

Figure 5.4 shows the system throughput in the execution of the previous tests. The overhead induced by the virtualisation can be clearly seen in the difference in the asymptotic throughput. In this case, the overall system performance is reduced 13 % (from 0.011 jobs/second in Test1 to 0.0095 jobs/second in Test2). Also, it is interesting to note that in all cases the system needs the same number of jobs to achieve half of the peak performance (see [110]).

5.4.2. Scheduling executions in cloud federations

The intention now is to evaluate the suitability of the GWcloud drivers for scheduling calculations among cloud providers following the early-binding approach. Moreover, the objective also is to demonstrate its features: the discovering and monitoring of providers, the scheduling based on several policies, and overall, the possibility of using the virtual environment that the legacy application requires.

For this purpose the EGI FedCloud infrastructure was used to perform real calculations with SAS. As that of May 2015, the infrastructure is considered in production and it counts on more than 24 providers, which offer more than 15,000 cores under the *fedcloud.egi.eu* virtual organisation. Presumably, the overheads obtained in this experiment should be similar to previous Test3 or Test4, plus the contextualisation and configuration time. Additionally, cloud providers should be much reliable and

Table 5.3: FedCloud IaaS providers actually used in experiments. Sites that do not accomplish minimal requirements (image, OCCI 1.1, encrypted endpoint) are omitted. Additionally, the technology used in every resource is also shown.

Provider	***resource_tpl#***			**Max.**	
(OCCI endpoint)	*ID*	GB	**Hyp.**	**Cores**	**VIM**
https://carach5.ics.muni.cz:11443	small	2	Xen	960	OpenNebula
https://cloud.cesga.es:3202	small	2	Xen	432	OpenNebula
https://controller.ceta-ciemat.es:8787	m1-small	2	KVM	224	OpenStack
https://egi-cloud.pd.infn.it:8787	m1-small	2	KVM	96	OpenStack
https://fc-one.i3m.upv.es:11443	small	1	KVM	16	OpenNebula
https://fsd-cloud.zam.kfa-juelich.de:8787	small	1	KVM	600	OpenStack
https://head.cloud.cyfronet.pl:8787	m1-small	2	KVM	1,600	OpenStack
https://occi.cloud.gwdg.de:3100	small	1	KVM	768	OpenNebula
https://occi.nebula.finki.ukim.mk:443	small	1	KVM	360	OpenNebula
https://nebula-server-01.ct.infn.it:9000	small	1	KVM	600	OpenNebula
https://nova2.ui.savba.sk:8787	m1-small	2	KVM	408	OpenStack
https://prisma-cloud.ba.infn.it:8787	small	1	KVM	600	OpenStack
https://sbgcloud.in2p3.fr:8787	small	1	KVM	447	OpenStack
https://stack-server-01.ct.infn.it:8787	m1-small	2	KVM	600	OpenStack

available than grid sites. However, these two considerations are not accurate, as was demonstrated through the following experiment.

5.4.2.1. Test-bed and scheduling configuration

A virtual machine with one Xeon X5560 (2.8 GHz) core and 4 GB of RAM was configured with GridWay 5.14 and GWcloud drivers to act as *client host*. The appropriation of resources was unrestricted per every provider and GridWay was allowed managing 500 jobs at the same time. The banning of providers was enabled with the default configuration (providers are banned for a maximum of one hour).

The latest SAS 14.0.0 release was selected to be distributed into cloud. As commented at the beginning of this Section, SAS supports few OS, in this case only four Linux releases, from which, the Ubuntu 14.04 is the only one available in FedCloud. Consequently, the GWcloud ID driver was configured to dynamically filter the providers discovered in FedCloud. First, the Basic Ubuntu Server 14.04 LTS image was selected from the *appdb.egi.eu* repository. Therefore, it only notifies the IaaS providers that publish at least one *os_tpl* with the corresponding image identifier[3] as description. Subsequently, the hardware templates (*resource_tpl*) are constrained to the ones offering one core and a minimum of 1 GB of RAM. Note that the driver automatically filters resources supporting the 1.1 release of OCCI offered through an encrypted endpoint.

To demonstrate the compatibility with legacy applications, the DRMAA library described in Section 4.3 was used to manage the calculation. It managed 500 jobs, which contain a script that installs and configures the SAS and HEAsoft software and then performs a default complete analysis (camera, spectrometers and optical monitor) based on the ODF provided. Obviously, ODF files can be directly downloaded from the XSA repository by the job, but to preserve the comparison with other experiments performed through this thesis, all the files were transferred through common protocols from the *client host*, in this case using GASS. (As the experiment is an example, only the observation number 0144090201 was used, but always is transferred

[3] https://appdb.egi.eu/store/vo/image/de355bfb-5781-5b0c-9ccd-9bd3d0d2be06

Table 5.4: Direct job executions in VMs deployed in EGI FedCloud.

Provider	Status	Total	OK	F	Err	Hung
https://carach5.ics.muni.cz:11443	OK	807	311	496	496	0
https://cloud.cesga.es:3202	up/saturated	10	0	10	10	0
https://controller.ceta-ciemat.es:8787	OK	244	164	80	80	0
https://egi-cloud.pd.infn.it:8787	up/saturated	10	0	10	10	0
https://fc-one.i3m.upv.es:11443	OK	21	10	11	11	0
https://fsd-cloud.zam.kfa-juelich.de:8787	up/saturated	31	0	31	31	0
https://head.cloud.cyfronet.pl:8787	up/saturated	10	0	10	10	0
https://occi.cloud.gwdg.de:3100	up/saturated	10	0	10	10	0
https://occi.nebula.finki.ukim.mk:443	misconfigured	14	0	14	10	4
https://nebula-server-01.ct.infn.it:9000	up/saturated	12	0	12	12	0
https://nova2.ui.savba.sk:8787	misconfigured	14	0	14	13	1
https://prisma-cloud.ba.infn.it:8787	up/saturated	13	0	13	13	0
https://sbgcloud.in2p3.fr:8787	up/saturated	10	0	10	10	0
https://stack-server-01.ct.infn.it:8787	misconfigured	19	0	19	9	10
Total		1225	485	740	725	15

as if were different inputs). Thus, SAS and HEAsoft software are transferred as inputs of the job as well as the ODF and CCF files. Their decompressed sizes were already commented at the beginning of the section. The compressed size of the former ones results in 1.3 GB to transfer. Outputs, ODF and CCF files are 339, 582 and 600 MB respectively, which take up 596 MB compressed for the staging processes.

Therefore, the configuration and the calculations described are similar to the ones performed in Section 4.5, allowing the comparison. However, performance thresholds have not set for jobs and the suspension timeout was selected deliberately higher (15 minutes) to avoid discarding jobs due slow building and contextualisation of the VM.

5.4.2.2. Results

The number of theoretically suitable providers (14) are around half of the available ones (24) in FedCloud. This is mainly due to the fact that the required *os_tpl* is not deployed in every site.

Table 5.4 shows similar information than the Table 4.3, but adapted to the cloud behaviour. The *status* of providers, the jobs successfully executed (OK) and the total number of failed jobs (F) maintain their meaning. However, the source of failures is different. "Err" stands now for the impossibility of creating the VM, i.e. the quotas or the maximum occupation for the provider have been reached. As in grid, the number of submitted jobs to every provider demonstrates a correct distribution in the infrastructure, but their real availability is far from considering the infrastructure as reliable. Thus, the only way to know if a provider can instantiate a VM is periodically testing its creation thanks to the banning feature. On the other hand, "Hung" indicates that a VM seems started, but neither ends and nor returns outputs. Therefore these jobs are lost, being the reason of achieving a number of completed jobs under 500.

Time spent by the complete test on cloud was 8 h 27 m 26 s. Taking into account that one analysis expends 31 minutes on a node with a Xeon X5365 processor and with SAS software previously configured, and that the number of successfully ended jobs are 485, the distribution of the calculation among cloud resources is ~ 29.6 times faster than the sequential execution. This is a great result if it is considered that only three providers have contributed to the calculation. In any case, the objective of evaluating

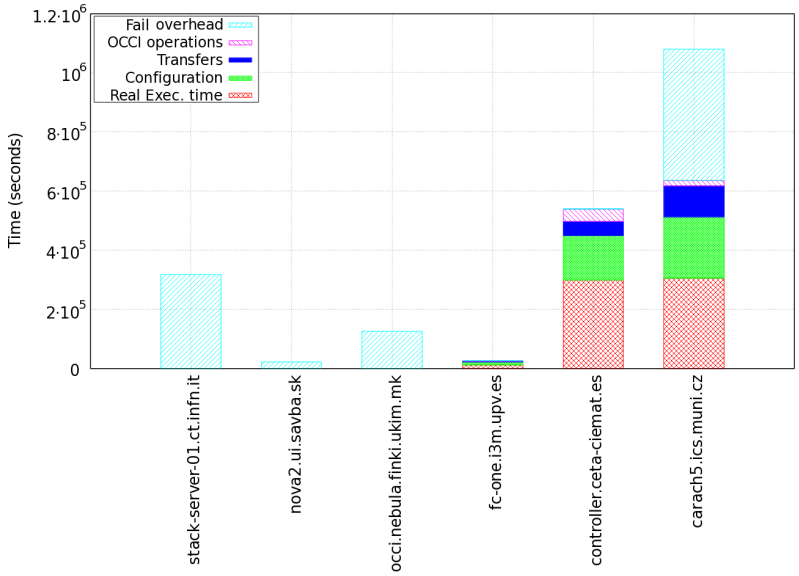


Figure 5.5: Accumulated overheads classified according to cloud providers that have been significantly contributed to the calculation. Additionally, the impact of hung VMs is included.

the suitability of the GWcloud drivers for enabling GridWay with scheduling capacities in cloud environments is achieved.

Other issue is the overheads generated. Besides of the ones generated by failed operations, mainly due to "Err", the accumulated time spent by successfully OCCI operations was 64 h 14 m 47 s, while the configuration of software was 99 h 39 m 19 s, the time in staging was 42 h 20 m 55 s and the effective execution time was 167 h 12 m 47 s. Thus, the overhead for ended jobs constitutes the $\sim 55.2\,\%$ of the calculation. These measurements do not include the time wasted by hung VMs.

To allow the comparison of these results with the performance obtained in grids following the early-binding approach, a picture similar to Figure 4.2 is shown, but now the "intrinsic overhead" is split into the ones resulted from OCCI operations, the configuration of SAS and transfers. As it can be seen in Figure 5.5, the impact of failures on the accumulated overhead is similar to grid as well as the other overheads are comparable to the waiting times due to grid queues and middleware. It is clear that the latter assessment only fits to this experiment and other executions relying on scientific software with fewer requirements will achieve better results. However, it also illustrates the poor performance of executing short jobs when the customisation of the virtual workspace is performed, being the capacity of customisation one of the main features achieved. Therefore, the methodology presented in this chapter is only suitable for accomplishing long jobs.

5.5. Conclusions

In this chapter, a new methodology to straightforwardly schedule and deploy virtual machines in current cloud federations has been presented. The approach supports the direct execution of jobs, being compatible with legacy applications. Moreover, it

allows users to select and build customised virtual environments for these applications. The system is suitable for long jobs such as bag of tasks (BoTs), it is able to set up ephemeral services on-demand and to provision pilot jobs as will be shown in Chapters 8 and 11. These capacities have been demonstrated adapting it to a large sized application with specific requirements and dependencies, obtaining convincing results.

However, this solution presents several drawbacks like a limited use of the potential benefits offered by the virtualisation technology (e.g. server consolidation), the uncontrollable overheads during the VM creation, and the excessive transfer of common software and/or input files. These issues make it unfeasible for short calculations or its direct utilisation in pay-per-use clouds.

Moreover, last experiments have shown that the IS deployed in cloud lack of characterisation, as happened with grid. The only way to know the quotas established, the overheads and the hardware really provisioned is continuously testing the creation of VMs in every cloud provider. However, the estimation of the duration of jobs, the overheads and the influence of failures could be performed by means of some techniques such as those described in Subsection 2.3.3 or 2.4.4. Therefore, specific Workload Scheduling algorithms should be developed to improve the behaviour of the solution if the early-binding approach is followed. Those scheduling mechanisms should be similar to the ones developed in self-schedulers for grid environments. This aspect is tackled in the following Chapter 6.

Chapter 6

Deploying Self-scheduling Techniques

6.1. Introduction

In this chapter, the objective is to evaluate the effectiveness of advanced Workload Scheduling algorithms following the early-binding approach. Dynamic self-schedulers are the most feasible tools for this purpose because they are able to adapt the calculation according to the estimation of the infrastructure status, but following their own criteria and disregarding the information from the IS when needed. In this sense, the mouldability supported by simple Monte Carlo (MC) codes makes them more suitable for the study than the parameter sweep ones.

Therefore, a self-scheduling framework devoted to improve the execution of MC codes is deployed. The system was firstly presented in [227] and substantially improved through years [109, 67]. Thus, the framework introduced in this chapter is a complete re-implementation that enhances its stability and robustness, while providing new capabilities and reducing the overloads. It now employs a set of advanced techniques and dynamic information gathering in order to optimise the size and distribution of jobs into heterogeneous and changing environments. In this sense, the framework supports an improved version of DyTSS (*Dynamic Trapezoid Self-Scheduling*) algorithm to perform the adaptive scheduling of MC applications. DyTSS requires knowing the theoretical performance of the infrastructure for computing a concrete calculation before assigning a number of samples to be computed by a certain resource. This performance is estimated following a mathematical model that is continuously fitted with the benchmarking information retrieved from ending jobs.

The system is properly explained though the following section and its behaviour is studied by executing several MC applications with diverse requirements. Furthermore, the improvements achieved are compared to the regular Job Scheduling performed by RBs.

6.2. Resilient executions of MC codes

To make use of the self-scheduling framework, users must fill only a template that includes typical statements (main executable, input and output location...), but also

the number of desired samples, and the minimum and maximum samples for every submitted job. The system will use these data to perform a resilient execution of the application on dynamic, unreliable and heterogeneous distributed resources.

To better understand the processes performed by the self-scheduler the following subsections are organised following the three loop phases stated in Subsection 2.4.4, this is: characterisation; Workload Scheduling algorithm; and, submission, monitoring and accounting.

6.2.1. Characterisation

The framework supports by default two characterisation modes: it estimates the available number, power and bandwidth of the slots offered by the infrastructure at a certain time, as well as the computational needs of the application. The main difference among other frameworks is how those parameters are estimated. In particular, the system:

- Benchmarks every provider by means of submitting a testing job that measures the CPU performance in manageable units (whetstones [254]) as well as the bandwidth.
- Then, averages the aforementioned benchmarks with every real execution of the real application, taking also into account the time needed for staging its input and output files.
- Estimates also the slot availability and reliability of every site averaging the queue times in remote queues, the number of failed attempts (suspension timeouts) and the number of failed jobs.

It is noteworthy to mention that not only the number of slots is variable, even the hardware provided by each resource is because it can be composed by different kinds of nodes. Therefore, the characterisation is achieved relying on the following innovative set of mechanisms that are focused on MC calculations.

Application profiling and resource benchmarking

The operation with a new MC code starts analysing its computational needs. To do so, it is executed on a set of known grid sites progressively growing the number of samples, so its requirements in terms of CPU consumption and data movement can be profiled. The obtained parameters are the *constant effort* C_{eff} (the effort necessary to execute the constant part of the application, like compiling code, input pre-processing, output post-processing, etc.), and *sample effort* S_{eff} (the effort necessary to simulate a single *sample*). These efforts are measured in the performance units provided by any general-purpose CPU benchmark tool to increase the suitability of the solution for a wide range of computational requirements. For this purpose, Whetstone [254] was selected.

Obviously, sites composing the distributed grid infrastructure are benchmarked too with the previous procedure. However, the data related to a provider are its *performance* (P) and its *bandwidth* (BW). To obtain the information, the benchmark tool is executed and a big file is copied. Additionally, an analysis of the job log allows obtaining the waiting time at the remote queues (Q). This analysis is performed when

a new site is detected, or even when a known one seems that it has been modified (e.g., it is detected with different CPU or memory capacities).

Simplifying the Equation 2.1, these parameters are used to estimate the turnaround time (T) of executing an arbitrary number of simulations (s) of a given application on any remote resource ($r_j \in R$), which calculation requires and generates certain data (D) to transfer:

$$\begin{aligned} T(r_j, s) &= T_{sched}(r_j) + T_{xfer}(D) + T_{exec}(s, r_j) \\ &\simeq Q_{r_j} + \frac{D}{BW_{r_j}} + \frac{C_{eff} + s \cdot S_{eff}}{P_{r_j}} \end{aligned} \tag{6.1}$$

However, this model is unreliable if benchmarking parameters are not updated through the time. Thus, every successful execution of a MC task is analysed to recalculate them, so knowledge about the infrastructure is enhanced in real time with minimum computational effort. The information about past executions is periodically compiled in the submission and accounting phase, so their behaviour can be estimated in any time.

Therefore, as the accurateness of the model is preserved through the changes in the availability of resources, it can be used by the specialised Workload Scheduling algorithm to determine the global performance of the infrastructure and to consequently adapt the number of samples (s) to submit and where do it.

Slot availability and reliability

The determination of the exact slot availability and reliability of every provider is obtained by their progressive overload with real executions, employing the simple yet effective mechanism based on maintaining a 20 % of queued jobs over the running ones on every site. This procedure was performed in the submission and accounting phase and it can be considered as a replication mechanism. However, the objective is to improve the characterisation of providers, unlike other replication approaches which are using this technique as the only way to reduce makespan. For this reason, replication is limited with diverse techniques as will be explained in the corresponding Section 6.2.3.

Therefore, the interest now is to describe the advantages of this mechanism to improve the characterisation of providers. For this purpose the process has to be somewhat introduced. First, a single job is submitted, when it starts running, this 20 % indicates to submit another one. This overload continues until the number of running jobs stops growing, indicating the real number of available resources on that site. Thus, if this number grows or is reduced, the framework will detect this change during the accounting phase, which modifies the subsequent characterisation and then is used by the scheduling algorithm. Therefore, the framework is not taking into account the *available slots* shown at the IS due to that number is not the real volume available for a given user belonging to a certain VO.

Moreover, profiling is also retrieved in every job execution and compared with the stored ones. Therefore, the approach is to benchmark the performance of the provider as well as the reliability of the infrastructure. Obviously, samples calculated are subtracted from remaining ones, having influence on the next sample distribution performed by the scheduling algorithm.

Algorithm 1: DyTSS: *Dynamic Trapezoid Self-Scheduling.*

Require: $R \equiv [r_{max}...r_{min}]$ **(list of estimated available resources)**
Require: L **(minimum sample-chunk)**
Require: M **(maximum sample-chunk)**
Require: S **(number of required samples)**
$task\_list \leftarrow \{\emptyset\}$
for $r_j \in R$ **do**
  $task\_list \leftarrow task\_list \cup \{(r_j, L)\}$
$free\_res\_list \leftarrow R$
while $S > 0$ **do**
  for $i = 1 \rightarrow 100$ **do**
    $old\_task\_list \leftarrow task\_list$
    $n_{1/2} \leftarrow linear\_fit(old\_task\_list)$
    $F \leftarrow S/4 \cdot n_{1/2}$
    $task\_list \leftarrow \{\emptyset\}$
    for $r_j \in R$ **do**
      $f_{rel} \leftarrow T(r_{max}, L)/T(r_j, L)$
      $s \leftarrow minimum(M, F \cdot f_{rel} + L)$
      $task\_list \leftarrow task\_list + \{(r_j, s)\}$
    if $task\_list \equiv old\_task\_list$ **then break;**
  for $r_j \in free\_res\_list$ **do**
    $submit((r_j, s_j) \in task\_list)$
  $free\_res\_list \leftarrow \{\emptyset\}$
  for $end(s_j), (r_j, s_j) \in task\_list$ **do**
    if $OK(s_j)$ **then**
      $task\_list \leftarrow task\_list - \{(r_j, s_j)\}$
      $S \leftarrow S - \{s_j\}$
      $free\_res\_list \leftarrow free\_res\_list \cup \{r_j\}$
  for $new(r)$ **do**
    if $T(r_{min}) > T(r)$ **then**
      $R \leftarrow R \cup \{r\}$
      $free\_res\_list \leftarrow free\_res\_list \cup \{r\}$

6.2.2. Adaptive sample-based algorithm

DyTSS [227] is a kind of loop-based algorithm [20] that differentiates from other approaches such as TSS [255] or GTSS [108] in its dynamic nature and in its focus on managing MC codes. The enhanced version of DyTSS used in this thesis is described in Figure 1. It shows the pseudo-code resultant of observing more properly the mathematical basis stated in [227] and purging the functions without relation to the algorithm itself. Moreover, the main advance is the inclusion of L and M limits as external configuration parameters. With them, users can adjust better its behaviour on certain infrastructures, as well as configure the overload of the replication performed during the submission and accounting phase, being of much importance the value set for L, as will be explained in Section 6.3.

Dynamic algorithms usually need to continuously calculate the suitable execution time in every provider to fit the workload partitioning to the estimated status of the infrastructure. For this reason it is usually necessary to benchmark the infrastructure

performance and to profile the application. Some general-purpose algorithms [216], and even ones devoted to MC [207], are based on calculating the minimal real execution time in every resource to reduce the overhead percentage of every job to a certain threshold. All of them can be benefited from the accurate estimation procedures mentioned through last subsection.

With DyTSS the approach is completely different. The minimum (L) and the maximum chunk-size (M) are preset by the user. These values can be previously calculated taking into account the mentioned overheads, but the algorithm will never modify those values. The approach of DyTSS is to reduce overheads adjusting first executions as much as possible to M, but straighten turnarounds. Then, it progressively decreases the chunk size to L, overcoming the influence of failed jobs in makespan.

The algorithm calculates the number of samples (s_j) to submit to every available resource (r_j) belonging to a characterised infrastructure (R). For this purpose, the current power of the infrastructure is estimated by linear regression of the ordered performances from (r_j, s_j) pairs. The ordinate in the origin ($n_{1/2}$) of the obtained straight-line corresponds to the half-performance of the infrastructure for this distribution of tasks [110]. This value determines the variable component (F) of the maximum chunk size ($F + L <= M$) for the current loop stage. As commented, L and M are constant, and consequently the number of samples is always within the interval ($L \cdots M$), but the turnaround will be different for every match (r_j, s_j). Thus, the procedure is repeated until no improvement is obtained for the sample distribution.

Note that (F) should decrease with the number of samples (S) as the simulation is being accomplished (end(s_j)) if not a new more powerful (new(r)) site is discovered. This assures cutting final execution tail due to remaining and standalone jobs.

6.2.3. Submission, monitoring and accounting

To perform the low level steps of the Job Scheduling, i.e. job management and resource discovering, the GridWay [10] meta-scheduler has been employed. This is principally motivated by its better performance [135], its failure detection mechanisms and its accessible monitoring and logging. These features are fundamental to build reactive algorithms such as DyTSS. For doing this, the framework relies on GridWay to properly manage jobs as well as to obtain the contact list of the sites belonging to the infrastructure, but disregarding its other functionalities such as the site statistics, automatic banning failing sites or even any other information shown for the sites (CPU power, queues, etc.), which is compiled from the IS.

Therefore, GridWay only acts as an efficient dispatcher. The framework delegates the *computation* to GridWay, building jobs through the DRMAA API. (Note that the framework uses DRMAA, but it does not support DRMAA applications. MC codes have to be adapted to the specific template of the framework). Additionally the system uses CLI to monitor the jobs.

Replication

The main objective of this technique is the properly characterisation of providers, as was commented in Subsection 6.2.1. Collaterally, there is a reduction of the execution makespan. By having some queued jobs at particular site, chances to start their

execution when there is a free slot increase. However, the intention is not disrupt the progressive Workload Division and matchmaking performed by DyTSS. Then, only L samples are calculated by replicated jobs.

With this approach, an optimal number of jobs will be created for every site, while keeping the number of replicas low: if the number of running jobs is reduced, the framework detects it and will consequently reduce the number of waiting jobs, cancelling those replicas still waiting so that no CPU time is lost. If the number of running jobs increases, the framework will create more jobs until reaching the 20 % of excess again.

It is important to mention that the difference between the proposed overload and pure replication algorithms is subtle, but highly relevant. In both approaches, the number of jobs submitted is greater than what is needed, but their behaviour is different. In the case of replication, the spare jobs running are cancelled, thus losing the computational effort put into their execution. However, in the case of DyTSS (given that in MC codes all results are equally valid) all the results are employed in the simulation. The only non-useful jobs are those being executed when the desired number of samples is reached. Moreover, as the execution time of these jobs is limited by L, CPU wasted is minimal.

Thanks to configuring L, users can obtain better performance with a controlled replication, while DyTSS takes advantage of the effective characterisation without abusing of the resource usage.

Accounting

Every turnaround is compiled measuring the time needed for staging input and output files. Additionally, the average queue times in remote queues, the number of failed attempts (suspension timeouts) and the number of failed jobs are compiled. Although the GridWay accounting performs similar procedures, it is not suitable for the framework operation due to it compiles the information in bulk. Additionally, application profiling and benchmarks have to be included. Moreover, job logs are processed to obtain more information.

For this purpose a relational database is created to store all the processed information regarding the remote execution of jobs. This is crucial to ensure persistence of characterisation among executions, which is necessary to build adaptive algorithms based on the behaviour and characteristics of the infrastructure along the time, overcoming slowdowns and hang ups.

6.3. Experimental evaluation

To measure the suitability of the self-scheduling approach presented in this chapter, a large comparison was performed: six applications from different areas of knowledge and computational needs were used for this task, namely BEAMnrc, FAFNER2, FLUKA, Nagano, Grif and FastDEP. Their complete description are available through the Appendix B. The execution of these applications were performed on the self-scheduling framework, an out-of-the-box installation of GridWay, and a remote WMS provided by the infrastructure.

The intention is to evaluate the performance gain of self-schedulers over traditional approaches relying on RBs. The inclusion of WMS in this comparative is motivated by its extensive usage, becoming the *de facto* standard in many experiments based on

Table 6.1: Scalability of the self-scheduling approach: walltime factor when problem size is increased. N_0 represents the size of the first experiment.

Code	N_0 (samples)	Walltime N_0 (seconds)	$Walltime_{a \cdot N_0}$ / $Walltime_{N_0}$ $a = 25$	$a = 50$	$a = 75$	$a = 100$
BEAMnrc	$1 \cdot 10^6$	1070	1.52	3.24	2.79	5.80
FAFNER2	$5 \cdot 10^3$	1235	4.83	6.66	5.73	11.07
FLUKA$^{(*)}$	$1 \cdot 10^5$	3515	4.89	8.45	15.13	16.69
Nagano	$2 \cdot 10^5$	3066	7.56	10.00	21.45	49.88
Grif	$2 \cdot 10^5$	6984	15.03	26.84	17.46	30.75
FastDEP	$3 \cdot 10^2$	9564	6.42	17.70	48.06	76.46

$^{(*)}$ 1 sample = 100 primary particles.

MC calculations. GridWay is included not only to be compared with WMS but also to serve as a reference to improvements achieved: as the self-scheduling framework runs on top of GridWay, this comparison allows determining how much of the performance gain is due to GridWay and how much to self-scheduling. Then, this comparison is not only useful for determining which of these approaches is more suitable for MC codes, this allows knowing how much self-scheduling overcomes early-binding issues.

6.3.1. Test bed and simulations

The comparison between the self-scheduling and RB approaches was accomplished on EGI. This is so because performing the calculations on a production infrastructure and not on a controlled environment allows demonstrating how the framework is able to overcome the issues that usually happen and are inherent to any grid execution [1]. This differs from the vast majority of studies in the area, where controlled, idealised or even virtual environments are employed.

Three machines configured with EMI-UI 2.0.2 middleware were set up. The first one was used for submitting jobs to a remote WMS endpoint; the latter two were configured with GridWay 5.14; and one of them is used by the self-scheduler. The WMS endpoint was chosen randomly at the beginning of every calculation among the available ones in the EGI infrastructure. No limits on the number of running jobs or any other parameter were set on WMS, using the default resources as provided by the remote service.

However, GridWay was configured with a maximum number of 50 running jobs and a scheduling interval of 25 seconds. This number of jobs allows testing the feasibility of the proposed approaches. As the self-scheduler runs on top of a GridWay installation, it has been configured with these same limits to avoid any distortion. It is important to bear in mind that, with this particular configuration, the number of resources employed at a given moment is potentially much higher with WMS than with both GridWay-based alternatives. Therefore, any modification of those limits would actually benefit the presented work compared with WMS.

On the other hand, applications had to be adapted to the three systems compared. For this purpose six scripts that make use of the WMS remote CLI have been developed for every application. The adaptation to GridWay was straightforwardly performed with the DRMAA producer-consumer library described in Section 4.3. Finally, templates required by the self-scheduler have been created.

The six applications have been executed with a set of 5 input sizes, so 30 different problems were carried out. For every code, an initial size (N_0) was selected that progressively grows ($a \cdot N_0$, $a \in [1, 25, 50, 75, 100]$). As was previously commented,

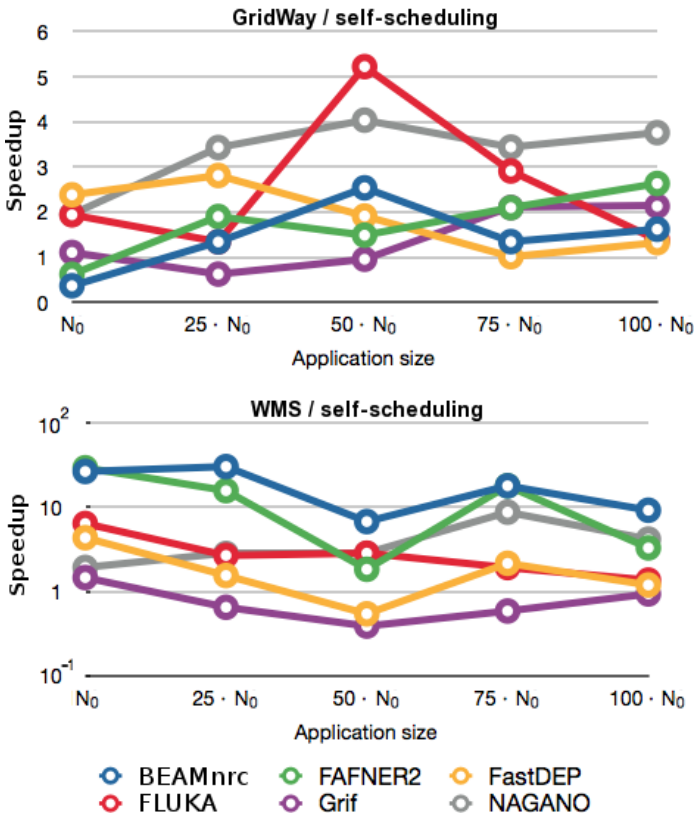


Figure 6.1: Speedup obtained with respect to GridWay (top) and WMS (bottom). Please note that the latter is in logarithmic scale.

in the characterisation phase the applications are profiled to be able to accurately predict their computational demands. As this information is previously known due to other executions, it has been also used to set the L parameter to the minimum number of samples that constraints the overhead of C_{eff} to 30 %, while M was unlimited. Whole data are summarised in the first columns of Table 6.4.

The division of the simulations into jobs has been done in the same way for WMS and GridWay: 50 jobs to execute N_0, and 125, 250, 375 and 500 jobs for $25 \cdot N_0$ to $100 \cdot N_0$. The number of samples to simulate in every job has been adjusted accordingly.

To avoid any distortion due to the infrastructure status or workload, the comparison was performed on a *competitive* way: every day at the same time, same problem on each platform (self-scheduler, WMS, and GridWay) was calculated.

6.3.2. Results

Improved execution time

Table 6.1 shows the walltime of the experiments performed with the self-scheduling framework. Its scalability is demonstrated because the execution time gradually grows as the size of the problem (a) is greatly increased. Thus it indicates an improved distribution of jobs and number of samples in the grid. It is important to remark

that, while the size of the problems varies in two orders of magnitude, their precise sizes have been chosen so they make sense from a physical and a computational point of view. For every application, at least the largest case ($a = 100$) corresponds to a real experiment.

Therefore, the makespan depends on the size of the problem to simulate, although the relationship is not directly proportional, and varies from every experiment. This is the reason for including the walltime factor in Table 6.1. In the smallest cases for BEAMnrc, FAFNER2 and FLUKA, the inferior limit L affects their execution time, as the number of samples submitted is greater than the real needed ones, so their execution takes longer. However, as the simulation size grows, this overload is reduced and the execution times should be closer to the expected ones. This aspect will be deeply explained later. On the other hand, an increment on the number of samples to simulate leads to a smaller increase on the total walltime because more jobs are distributed, and consequently, more middleware overheads related to job submission are overlapped.

The analysis of the proposed solutions has been completed with the comparison among GridWay, WMS and the self-scheduling approach. Fig. 6.1 and Table 6.2 compiles the results on two different ways. Fig. 6.1 shows the speedup obtained with self-scheduling, while Table 6.2 allows the comparison of walltimes. In general, the execution time with WMS is much larger than the others. The main WMS drawback is not its throughput on stable environments, but its lack of adaptability to dynamic ones: due to an unreliable error control, it is unable to detect hung or lost jobs in many cases. This is of importance because just a small group of jobs not being correctly executed will need to be resubmitted and will dramatically increase makespan.

On the other hand, the behaviour of WMS improves as the problem size increases. This is due to the greater number of resources allowed, which is limited to 50 in the case of the other tests, but unlimited in WMS as has been stated in Subsection 6.3.1. In the best scenario, this difference only leads to a superior performance over the self-scheduling approach of about 20 %, as it can be seen in four executions of Grif and one of FastDEP. This situation is however unlikely to happen, as the failure rate is about 9 %, and the submitted experiments are composed by 50 to 500 jobs.

For certain experiments, the employment of the self-scheduling approach instead of a standalone GridWay does not imply a significant improvement. As commented, when executing small experiments it simulates too many samples due to the inferior limit L, and consequently makespan is negatively affected. On the other side, if the number of jobs to execute is smaller or equal to the maximum limit established on GridWay, all of them should immediately start their execution. If they are all assigned to free and fast resources, the result is a fast and reliable execution. Of course, when the number of jobs to execute or their chunk-size is increased, the advanced self-scheduling techniques impressively reduce the makespan. Therefore, in large executions the overheads (queue time, data transfer and constant time of the application) are minimised by the self-scheduling framework and represent only a small fraction of the total wasted time in the experiments only performed by GridWay.

In conclusion, the proposed solution reduces the execution time to a third on average with respect to GridWay, being more than 30 times smaller on the best case with respect to WMS. On the other hand, self-scheduling offers important advantages besides its efficiency: an automatic and unattended Workload Division, Job Building, and a fully functional fault tolerant mechanism. However, the main achievement for the user's point of view is the speedup obtained in real simulation cases, for this

Table 6.2: Time saved by self-scheduling with respect to GridWay and WMS. (Speedup is shown in Fig 6.1).

Code	$walltime_{GridWay} - walltime_{self-scheduling}$				
	N_0	$25 \cdot N_0$	$50 \cdot N_0$	$75 \cdot N_0$	$100 \cdot N_0$
BEAMnrc	- 00 h 11 m 20 s	+ 00 h 09 m 10 s	+ 01 h 28 m 43 s	+ 00 h 17 m 05 s	+ 01 h 03 m 54 s
FAFNER2	- 00 h 07 m 49 s	+ 01 h 29 m 36 s	+ 01 h 06 m 34 s	+ 02 h 09 m 21 s	+ 06 h 11 m 59 s
FLUKA	+ 00 h 55 m 01 s	+ 01 h 37 m 21 s	+ 34 h 50 m 35 s	+ 28 h 12 m 57 s	+ 31 h 14 m 53 s
Nagano	+ 00 h 48 m 32 s	+ 15 h 38 m 45 s	+ 25 h 51 m 17 s	+ 44 h 34 m 28 s	+117 h 14 m 53 s
Grif	+ 00 h 12 m 06 s	- 11 h 15 m 21 s	- 02 h 27 m 19 s	+ 37 h 13 m 58 s	+ 67 h 25 m 02 s
FastDEP	+ 03 h 40 m 12 s	+ 30 h 51 m 18 s	+ 42 h 50 m 24 s	+ 01 h 16 m 36 s	+ 65 h 00 m 04 s

Code	$walltime_{WMS} - walltime_{self-scheduling}$				
	N_0	$25 \cdot N_0$	$50 \cdot N_0$	$75 \cdot N_0$	$100 \cdot N_0$
BEAMnrc	+ 07 h 34 m 47 s	+ 13 h 11 m 24 s	+ 05 h 33 m 44 s	+ 14 h 02 m 49 s	+ 14 h 10 m 37 s
FAFNER2	+ 09 h 43 m 10 s	+ 24 h 24 m 27 s	+ 01 h 56 m 54 s	+ 33 h 19 m 36 s	+ 08 h 47 m 43 s
FLUKA	+ 05 h 18 m 47 s	+ 08 h 06 m 34 s	+ 15 h 27 m 01 s	+ 13 h 55 m 57 s	+ 06 h 13 m 22 s
Nagano	+ 01 h 48 m 51 s	+ 11 h 54 m 41 s	+ 15 h 45 m 03 s	+141 h 45 m 42 s	+136 h 21 m 52 s
Grif	+ 00 h 55 m 50 s	- 10 h 12 m 29 s	- 31 h 27 m 10 s	- 13 h 40 m 47 s	- 03 h 06 m 14 s
FastDEP	+ 08 h 57 m 21 s	+ 09 h 39 m 31s	- 21 h 02 m 31 s	+150 h 38 m 58 s	+ 42 h 39 m 25 s

Table 6.3: Speedups obtained in real calculation cases ($a = 100$).

Code	**Serial execution time** Xeon E5620 2.4GHz (Launch date: 2012)	**Speedups in** $100 \cdot N_0$		
		self-scheduling	GridWay	WMS
BEAMnrc	002 d 00 h 12 m 37 s	27.97	17.29	3.03
FAFNER2	012 d 00 h 43 m 21 s	76.03	28.88	22.93
FLUKA	021 d 03 h 19 m 56 s	31.13	10.67	22.52
Nagano	070 d 12 h 53 m 04 s	39.85	10.60	9.47
Grif	196 d 18 h 13 m 20 s	79.16	37.16	83.50
FastDEP	023 d 11 h 24 m 51 s	2.77	2.10	2.29

purpose the Table 6.3 is shown. As the number of usable slots is limited to 50 for the GridWay-based approaches, higher values indicate that infrastructure is offering better resources than the hardware model.

Reliability

Regarding WMS reliability, performed experiments show a job failure rate of 1.5 %. Besides that, an additional 2 % was considered as done but returned an exit status different from zero (so had to be resubmitted), and nearly a 5 % of the jobs were cancelled by the user after staying for more than 24 hours on a *submitted* state but not starting their execution. All together, they represent approximately a 9 % of failed jobs. This value is consistent with past experiences [227] from different research teams [136]. It is important to regard that this failure rate is not only important by itself, but for the influence on makespan: if no replication mechanism or other control technique is employed, the failure or slowdown of a single job can have a direct influence on the overall walltime.

In the case of GridWay no jobs were incorrectly marked as ended, and all the failed ones were correctly resubmitted. However, GridWay itself hung five times during the execution of this set of applications, and only in two of them the job status could be recovered after restarting it. It did not also detect the missed work of a given remote site that was accepting as many jobs as possible but did not execute them, so it had to be manually banned. Additionally, about 0.5 % of the jobs remained in a wrong *active* state forever. In conclusion, while its performance regarding on fault tolerance is much better than WMS one, it does not provide the complete security that all the desired simulations will be seamlessly carried out, no matter the status of the infrastructure or the problems of a particular site.

During the experiments, the self-scheduling framework did not have to face any GridWay instability. However, due to its site profiling and analysis mechanism, no site had to be manually banned. Of course, some of the jobs were lost or assigned to unreliable resources, but the resilient design of the framework allowed overcoming these issues, by redistributing the workload and finally achieving the desired number of simulations on a completely unattended way.

Replication overload

In this work neither GridWay nor WMS impose an overload on the infrastructure in terms of additional samples or jobs. However, both approaches have a non-negligible failure rate, which was explained in previous subsections.

On the other hand, the self-scheduler framework described in this chapter uses

Table 6.4: Replication overload performed by the self-scheduling framework in terms of non-necessary submitted samples. It includes samples in either executed, waiting or cancelled jobs. Selected L is included to understand their influence on replication.

Code	**profiling** (*whetstones*)			**submitted / required simulations**				
	C_{eff}	S_{eff}	L	N_0	$25 \cdot N_0$	$50 \cdot N_0$	$75 \cdot N_0$	$100 \cdot N_0$
BEAMnrc	13034.10	0.89	44379	13622.68	248.82	125.10	105.37	199.65
FAFNER2	30370.02	1066.01	86	642.54	3795.77	38.25	21.84	43.21
FLUKA	975.60	93.66$^{(*)}$	32	59.42	6.21	6.15	5.67	7.98
Nagano	375.07	156.26	7	33.30	110.93	113.01	32.87	45.65
Grif	2779.38	435.88	19	86.81	12.18	4.41	13.67	0.3
FastDEP	81018.90	34670.25	7	69.33	9.31	4.64	2.86	1.13

$^{(*)}$ 1 sample = 100 primary particles.

replication, not really to directly improve makespan, but to characterise the providers. Note that, DyTSS algorithm is based on submitting a manageable number of samples and replication cannot exceed certain limit. To measure the overload introduced, Table 6.4 compiles the average overload generated in every experiment as a proportion of the needed samples to be calculated.

In general, smaller experiments (with lower a) generate a bigger overload rates. This is due again to the nature of the scheduling algorithm, which imposes the lower limit L on the task size to reduce the grid-related overheads and the influence of C_{eff} in the total execution time. As there are few samples, they are quickly distributed among resources and many jobs usually compute a number of samples close to L. As the replication mechanism also uses the limit L, it proportionally increases the number of samples submitted in smaller experiments. In contrast, when the experiment size is large, this low threshold is not reached for the regular jobs built by DyTSS until the calculation ends. Consequently, the overload rate is progressively reduced as the experiment size grows.

It is important to notice that not all of those jobs need to start their execution as the framework cancels them once it detects that is exceeding the established 20 % rate or it receives the notification of completion of the desired number of partial simulations. This way the wasted computation on the infrastructure is kept as reduced as possible while profiting from the fast execution time and robustness provided by DyTSS. Therefore, although the overload on the smaller cases is quite large, it is measured as a percentage of the submitted samples, not of the started jobs. This means that the CPU wasted is actually much reduced. For large experiments (i.e. the real case ones, $100 \cdot N_0$) the wasted computation remains under 10 %, what can be considered as an acceptable rate. This rate was measured compiling all the samples included into running jobs that were cancelled due to their uselessness at the end of calculation.

In any case, the lower limit L can be adjusted to balance the reduction of makespan and the overload generated in the infrastructure.

6.4. Conclusions

The main achievement of this chapter is the development of a new self-scheduling framework that enables users to perform resilient executions of their MC codes. However, the main objective was the evaluation of self-scheduling techniques for supporting Workload Scheduling algorithms while solving the early-binding issues. In this

sense, the framework deploys diverse mechanisms to characterise the infrastructure, implements an adaptive algorithm to fit the Workload Division to the status of every provider and properly controls the executions submitted. The result is a robust and fault tolerant system under any condition that overcomes problems both on local and remote resources. These facts are demonstrated through a comparative study against the regular performance achieved by RBs. The conclusions are summarised as follows:

- Self-scheduling techniques are able to improve the performance through the time using early-binding approaches, but not are suitable for calculations shorter than few hours.
- Improvements over other approaches are demonstrated, but they are variable. Although self-scheduling deal with the dynamism of the infrastructure, it is equally subject to the changes, being the exact performance unpredictable in any way.
- Self-scheduling frameworks are usually incompatible with legacy applications based on standardised APIs, but they also usually support skeleton facilities (see Subsection 2.3.6) to ease the adaption and to release developers to implement the *coordination* of jobs.

First two items are due to a deficient characterisation. Although the self-scheduling framework strongly improves this aspect, the complete forecast is not achievable. Despite the Workload Scheduling algorithm is designed to deal with this issue, overheads related to queues continue having a high weight in the computation. However, these aspects can lack importance if pilot jobs are used. Queues are overcome and the complete monitoring of provisioned resources should be achieved by pilots. Thus, with a pilot framework that allows the customisation of characterisation and supports stacking legacy third-party tools, the self-scheduler deployed in this chapter should properly improve the execution of MC codes. This goal is precisely one of the main objectives of this thesis and it is demonstrated in Chapter 11.

Part II

MULTILEVEL SCHEDULING WITH PILOT JOBS

Chapter 7

The GWpilot Framework

7.1. Introduction

As it has been demonstrated in Chapters 4 and 5, makespan of some calculations can be strongly reduced by following early-binding approaches on grids and clouds. The use of standardised APIs and RBs reduce the complexity of dealing with a huge volume of resources. The methodology for this purpose has been clearly stated. Additionally, a new technology has been deployed to enable resource brokering for distributed applications in clouds. Nevertheless, these brokering approaches only can achieve a limited performance due to the lack of characterisation of the infrastructures. High (and unexpected) overheads and the unknown availability of resources make early-binding only suitable for distributing long jobs. On the other hand, unpredictable failures increase the final makespan. To improve this behaviour, more advanced scheduling has to be performed in other aspects different to Job Scheduling. In this sense, self-schedulers as the one deployed in Chapter 6 implement their own characterisation to perform the Workload Division and Job Building. This approach can improve the reliability of calculations, but it is not enough. Characterisation continue being insufficient to fit the calculation to the real availability of resources and overheads are too high to assure an efficient profiting of them.

Therefore, it is necessary to make use of a late-binding approach to overcome these issues. However, this project cannot be undertaken with the current tools available because they do not allow customising their scheduling, or they lack of adaptability or compatibility, as it has been clarified through Section 2.5. To really support the legacy technologies and to enable users to manage the Multilevel Scheduling layers, a new framework has to be designed taking into account the objectives listed in Chapter 3.

GWpilot is designed to accomplish the requirements enumerated in Section 3.2 and to profit from the numerous GridWay advantages [256], while maintaining a simple design and implementing new functionalities that result in measurable and valuable improvements in several performance aspects of computing scientific applications. GWpilot acts as a GS-embedded pilot system, where pilots are included in the GridWay Host Pool as any other resource. Thus, user tasks will be included in the GridWay Job Pool and subsequently scheduled among pilots as if they were common grid jobs. Accordingly, the GridWay Scheduler module is used to perform the task-pilot and pilot-resource matchmaking. Therefore, the user fair-share and prioritisation capabilities can be incorporated into the pilot system if a careful design is

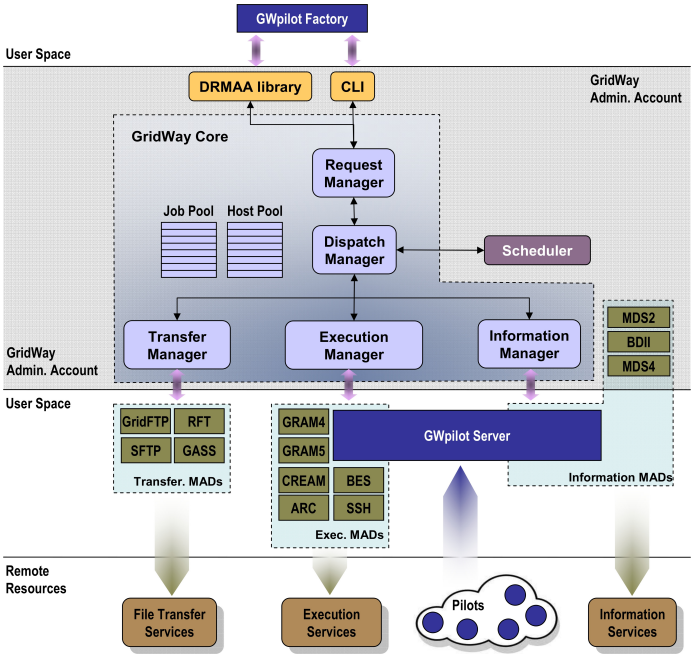


Figure 7.1: GWpilot components in GridWay architecture.

constructed. Similarly, the advanced scheduling capabilities of GridWay are improved by the proper characterisation and the online monitoring of resources and tasks that pilots provide, particularly the task migration, the checkpointing functionality and the matchmaking decisions based on statistical data obtained from previous executions, per user and per resource. Additionally, the incorporation of pilots makes new features possible, such as reservation, data-allocation awareness and caching. The user can easily implement specific scheduling policies if the necessary tools for declaring customised characteristics of tasks and of the pilot environments are available. In consequence, the new framework will allow users, developers and administrators to build every scheduling level (Application, Task Scheduling and Provisioning) accordingly to their specific needs in a unified and standardised way.

The main design and implementation of GWpilot that make all these features possible are described in this chapter. Additionally a functional and experimental study is carried out to compare the design and performance of the new framework with other two pilot systems, DIANE and DIRAC. On the other hand, the adaptability of legacy applications and the improved management of scheduling are deeply explained through Chapters 8 to 11.

7.2. Architecture

As it has been previously commented, GWpilot acts as a GS-embedded pilot system, where pilots are included in the GridWay Host Pool as any other resource. The GridWay Job Pool conceptually corresponds to the UTQ in the framework. Because of it, any user or developer can use the available interfaces from GridWay to manage tasks. As mentioned above, another benefit is to make use of GridWay Scheduler to

incorporate its algorithms to Workload Scheduling. The inclusion of every pilot in the GridWay Host Pool is the mechanism that allows this feature, which would have not been possible if the friendly interfaces had not been available or the language to describe resources had been difficult. In this sense, every resource (either pilot or the remote site) is described by means of unstructured label-value pairs. Users can easily inspect these values through CLI and set constraints into their descriptions of tasks. Subsequently, Scheduler calculates the best matches between the elements of two pools. In addition, the selection of resources for every pilot is visible for the users, who can influence on Provisioning. The advantages of this unified vision of the Workload Scheduling and Provisioning will be described later.

To preserve the performance and deploy-ability levels of GridWay, GWpilot must maintain its modular architecture [10]. Thus, the new elements of the system must be implemented like any other pluggable driver in the framework. For this reason, the system contains two main components in addition to the pilots: the GWpilot Server (GW PiS), which is implemented as a middleware access driver (MAD), and the GWpilot Factory (GW PiF), which acts as a common application at user-level and can be started by GW PiS. Subsequently, no communication will be performed between these components, and the system will run in a standalone way for submitting pilots wrapped inside grid jobs as needed. To perform this action, GW PiF uses other MADs (GRAM 2/4/5, ARC, CREAM, OGSA-BES, SSH, GWcloud), to obtain resources from multiple DCIs.

The code used for both components, PiS and PiF, is Python 2.4.3 compatible and, as any other MAD or user application within the framework, its installation is straightforward over a previously configured GridWay server. It only requires the copy of few source files to a basic GridWay installation and the deployment as Python module of a (third-party) generic open source HTTPS server framework[1]. GW PiS options are fully configured by editing the same line in the configuration file. These parameters are then propagated to GW PiF and, consequently, to the submitted pilots, so no more actions must be performed by inexpert users. Additionally, administrators can modify default parameters to tune general aspects of scheduling. On the other hand, MADs are generally executed in the user-space, which offers the following benefits: they can be independently instantiated by multiple users and they can directly use their grid proxy certificates for encoding communications. As GW PiF is executed by PiS, it profits from the same advantages.

Pilots communicate with the PiS through simple HTTP requests. A pilot will periodically pull PiS for a description of the task to run. This includes environment variables, the GridWay wrapper and the task options, as they were any remote GridWay job. Pilot translates these items and performs the necessary file staging to execute the wrapper. Subsequently, the wrapper performs the execution of the task as usual, assuring backward compatibility. Therefore, the stage-in and stage-out mechanism is established by the user as usual, i.e., by means of defining the source and destination of every file with local names or URIs through any standard grid protocol supported on the remote resource, e.g., GridFTP or SRM. On the other hand, HTTP requests are used to periodically advertise PiS about pilot characteristics and the statuses of tasks. In addition, tasks can communicate with pilots to arbitrarily include customised tags in its characteristics. As these items will be notified to PiS, and subsequently will be included in the Host Pool, the mechanism enables the personalised characterisation of the pilot.

[1] http://www.cherrypy.org

The implementation of pilots is fully compatible with Python 2.4.3 and its standard modules and can run, for example, on any Red Hat 5 minimal installation. Additionally, the code requires less than 1,000 lines and only uses approximately 40 KB and can be manually executed. Thus, GWpilot is potentially suitable for deployment on any currently distributed platform type such as the cloud, or even on a desktop computer, and for establishing network overlays among them, at least running in its insecure mode (i.e. without using grid certificates).

The design allows GWpilot to incorporate the following features that accomplish the requirements outlined in Subsection 3.2:

- Friendly user, administrator and developer interfaces from GridWay, such as the CLI, the submission mechanism based on templates, and the DRMAA and OGSA-BES standard interfaces. The latter also allows the use of remote commands through external implementations [257]. GridWay also provides DRMAA bindings to different languages such as Java, C/C++, Python and Perl.

- The security in communications and the file staging mechanisms, based on grid standards.

- The capacity to extend the pilot overlay to multiple DCIs, such as the grid, the cloud or even local resources, because PiF can use the current and future GridWay plug-ins (MADs) for Provisioning and pilots allow manual execution

- An easy and standalone deployment on a unique server, independent of other middleware instances in these DCIs. Currently, GridWay is available through .deb and .rpm packages and their local dependences (of grid middleware) are managed by official repositories (from Linux distributions and IGE[2]).

- Pilot communications based on minimal HTTP pull requests. Overheads are controlled by reducing the number and size of messages among system modules.

- Management of Workload Scheduling and Provisioning capabilities in a box. Accounting from both layers (users, tasks, pilots and resources) is available.

- Mechanisms to properly characterise resources at user-level, based on a simple description language compatible with the friendly interfaces provided.

- The possibility of performing a personalised scheduling by every user, because independent PiS and PiF can be instantiated for all of them. Additionally users can run manually PiFs or even substitute them by new developed ones to customise Provisioning.

- Management of multiple users and applications with fair-share and prioritisation policies. Potentially, PiS and PiF can be shared among users to share their managed pilot jobs and therefore, to carry on scientific production of a specific project or VO.

- Supporting the Workload and Provisioning layers with a default set of scheduling capabilities inherited from GridWay but powered by the pilot characterisation.

[2] http://www.ige-project.eu

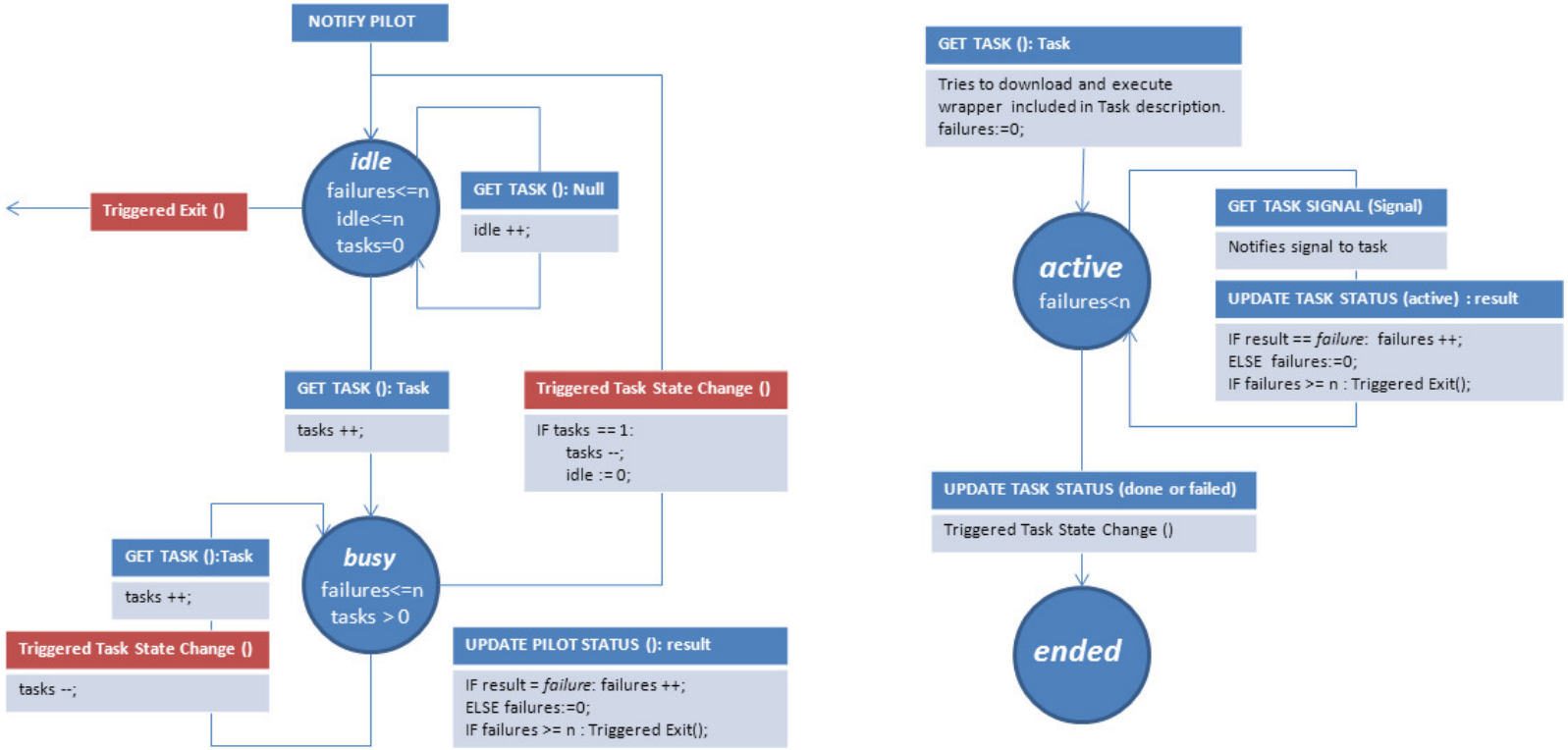


Figure 7.2: State machine diagrams representing pilot internal behaviour (left) and task management (right).

However, as mentioned in Section 3.2, overheads introduced constraints about what scheduling algorithms can be actually implemented on the pilot system. In particular, it is desirable that these overheads were predictable and even configurable. Consequently, beyond this overall description of GWpilot, a more complete explanation of its components is outlined below, paying special attention to the performance issues, but also to implementation details that make their advanced features possible.

7.2.1. Pilots

Communication with the PiS is a pull mechanism via simple HTTP requests from pilots. Due to the tag-based language used in GWpilot all the information interchanged is specific enough to be represented as a set of unstructured label-value pairs, although their significance is crucial for some of them, as will be explained below. Additionally, few request types are needed. Therefore, formatting the information in XML and using complex message protocols is unnecessary. Instead, the tags are set into variables of GET methods or received as plain text in their responses. Despite the similarity with the remote execution mechanisms and to differentiate from pure RPC protocols, the pilot requests are referred to as pilot operations throughout the remainder of this thesis.

The internal states are simple: a pilot can be *idle* or *busy*, and a task can be *active* or *ended* (*done* or *failed*). There are no pending tasks inside a pilot because whenever one is fetched from GW PiS, the pilot immediately tries to run it. As a consequence, there are only five operation types that the pilots request to the GW PiS:

- *Notify Pilot*: pilot advertises itself to PiS by sending its identification code and static characteristics.
- *Update Pilot Status*: when pilot considers itself enrolled to a specific PiS, it periodically sends its dynamic tags to that PiS.
- *Get Task*: when the pilot is in *idle* state, it asks PiS for a new task.
- *Update Task Status*: it is used to notify PiS of an *active* state when a task is going to be executed or the final state if the task has *ended*.
- *Get Task Signal*: it asks PiS for a POSIX signal to be passed to an *active* task. It is periodically performed when a task is being executed.

The last four operations are immediately triggered after a state change, thus reducing the turnaround. When no state change occurs, these operations are looped in a time interval with a limited number of retries, after which the pilot ends, either because PiS are not accessible or, in general cases, because there are no more tasks that can be assigned to the pilot. To better illustrate these state transitions two state machine diagrams extended with pseudo code are depicted in Figure 7.2.

Usually, parameters such as PiS port number, deactivation of SSL security, *pulling* frequency against the Server, and *number of retries* (T) are propagated by GWpilot configuration to pilots. However, pilots also allow manual execution, and they can be launched by customised factories relying on other GS or LRMS. Furthermore, these parameters do not only enable the communication with the Server. Retries determinate the Provisioning policy, i.e. how often pilots are discarded and interchanged by new ones. On the other hand, *pulling interval* (PI) is responsible of an important overhead component in task turnaround. Thanks to the simple design of pilot, this

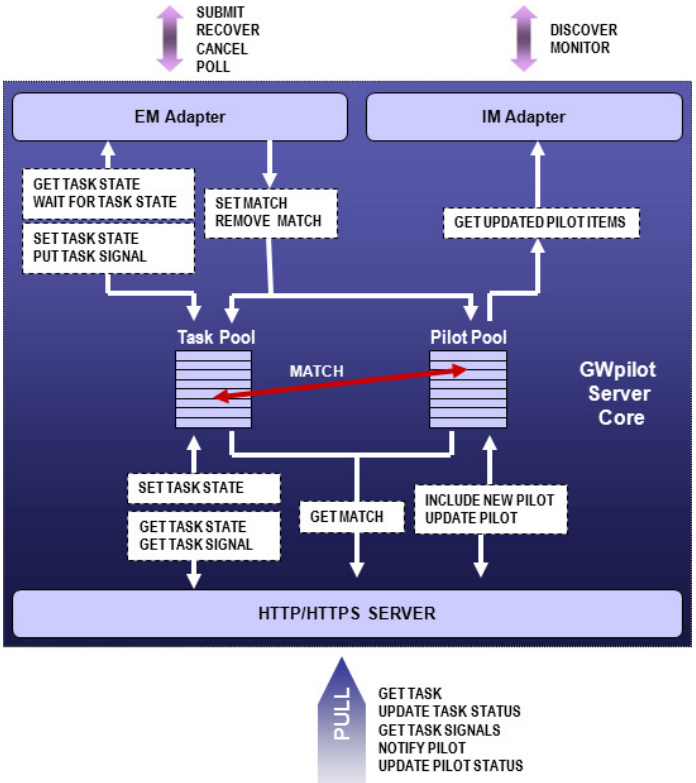


Figure 7.3: The GWpilot PiS internal modules, procedures, information workflow and its relation to external operations.

overhead is maintained roughly constant for every task, as will be demonstrated experimentally. The implications of both aspects will be explained through the following chapters, but especially in Chapter 11, where a statistical study is performed.

Secured communications are allowed using the delegated user proxy stored in the WN as a certificate for encrypting calls. Although SSL authentication is enough to distinguish the pilot owner, the pilot must identify itself with a unique code whenever an operation is performed against the Server. This identifier is formed by the WN hostname, the site name, the user name and a hash number, which is created when a pilot starts running and is maintained until its end. This method is performed to allow the proper consistency check of requested operations and to reject failing pilots. The exact identification of the remote and assigned resource will allow GWpilot to individually account performance statistics from each pilot. Such identification could be reutilised later if any other pilot is allocated in the same resource (this mechanism and its advantages will be explained in next subsection and Chapter 10, respectively).

Identification code is sent to PiS as any other property. Other tags have also special significance. $LRMS\_NAME =$ "*jobmanager-pilot*" is the type identifier that will differentiate pilots from other resources in the GridWay Host Pool; thus, it is always statically published when a pilot is enrolled in the system. On the other hand, queue tags dynamically show the number of available slots to accomplish tasks by the pilot. Scheduler will discard those pilots without free slots published in the Host Pool.

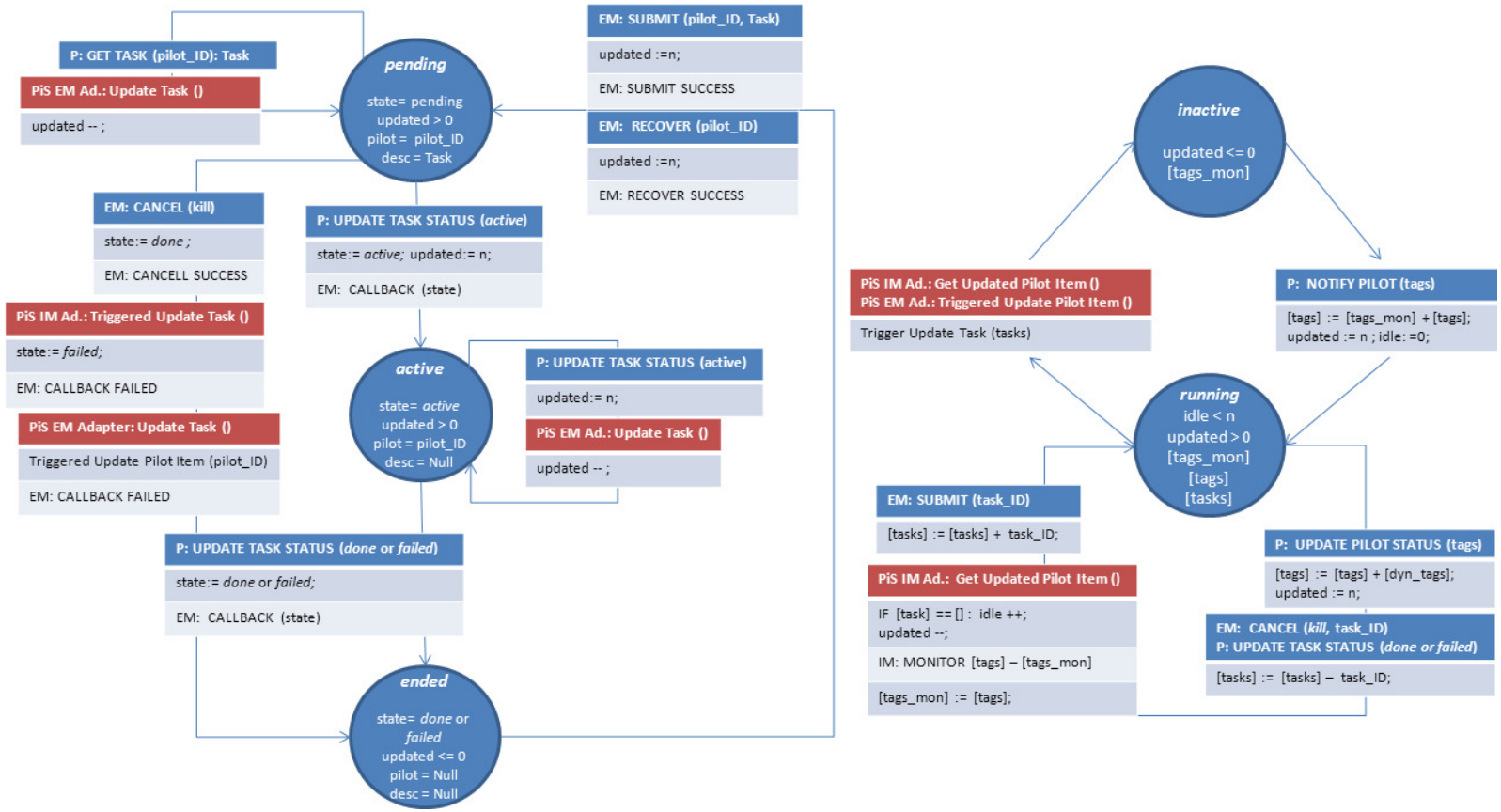


Figure 7.4: State machine diagrams representing the management of tasks (left) and pilots (right) by the GWpilot Server.

To assure the backward compatibility with previous GridWay developments, its remote execution wrapper is utilised in pilots. The execution of user task is actually performed by the wrapper in the specific directory where the wrapper is downloaded. Pilot receives the location (URI) of wrapper, the final locations of their output streams (its logs), and the necessary environment variables. These outputs are retrieved by the system, enabling the possibility of fine troubleshooting. The pilot also gets the complete description of the task because it is a parameter of the wrapper. Thus, the pilot is able to translate the description of every task to cache its executable, input, output and restart (for checkpointing) files. These files are retained according to their size and their local path. Subsequently, their MD5 are published. Additionally, pilot creates a named pipe on which the user task will can write pilot the customised tags. Subsequently, the pilot tries to run the wrapper and monitor it to trigger the status changes and to fill some specific tags about the execution (see Table 7.1). The wrapper also enables the capacity of checkpointing the task execution. The possibilities that those features offer are explained in Chapter 10.

7.2.2. The GWpilot Server (GW PiS)

The GW PiS is the most important module inside the GWpilot system, and its responsibilities go beyond simply putting GridWay in contact with pilot jobs. In this sense, PiS performs an active role by checking, filtering and caching operations from GridWay Core and pilots to improve the scalability and performance of the whole system. For this purpose, PiS must communicate with two GridWay managers: the Execution Manager (EM), that performs generic operations for tasks; and, the Information Manager (IM), that includes the characteristics of pilots in the GridWay Host Pool. This determinates the PiS internal design (see Figure 7.3), which is mainly composed of an EM and an IM adapter, the Task and Pilot Pool lists, and the implementation of necessary procedures embedded in a (third-party) generic open source HTTPS server framework[1] for Python.

PiS behaviour

The pilot and GridWay Core operations against PiS modules have associated internal procedures that imply changes in the task or pilot states and, consequently, trigger more operations to other modules and again, doing it outside PiS. Main procedures used by adapters and the information workflow are shown in Figure 7.3, while the state machine diagrams that depict the internal behaviour of the Server are depicted in Figure 7.4. In addition, there are other mandatory functions devoted to checking the validity of the task, the pilot or the match performed in an operation. It is relatively common that pilots perform invalid operations against the PiS HTTP server due to network overloads or cuts.

To better understand the behaviour of the Server, a simplified sequence of steps followed by the actors of the system to accomplish a task is described in Figure 7.5 which is explained as follows:

1. A pilot advertises itself by sending its identification code and static characteristics to the PiS. If the pilot is accepted, it begins *running* for the Server, which notifies the GridWay Core through the IM adapter. After that, the pilot periodically sends monitoring information, which is filtered by the PiS to only inform the IM about updated data.

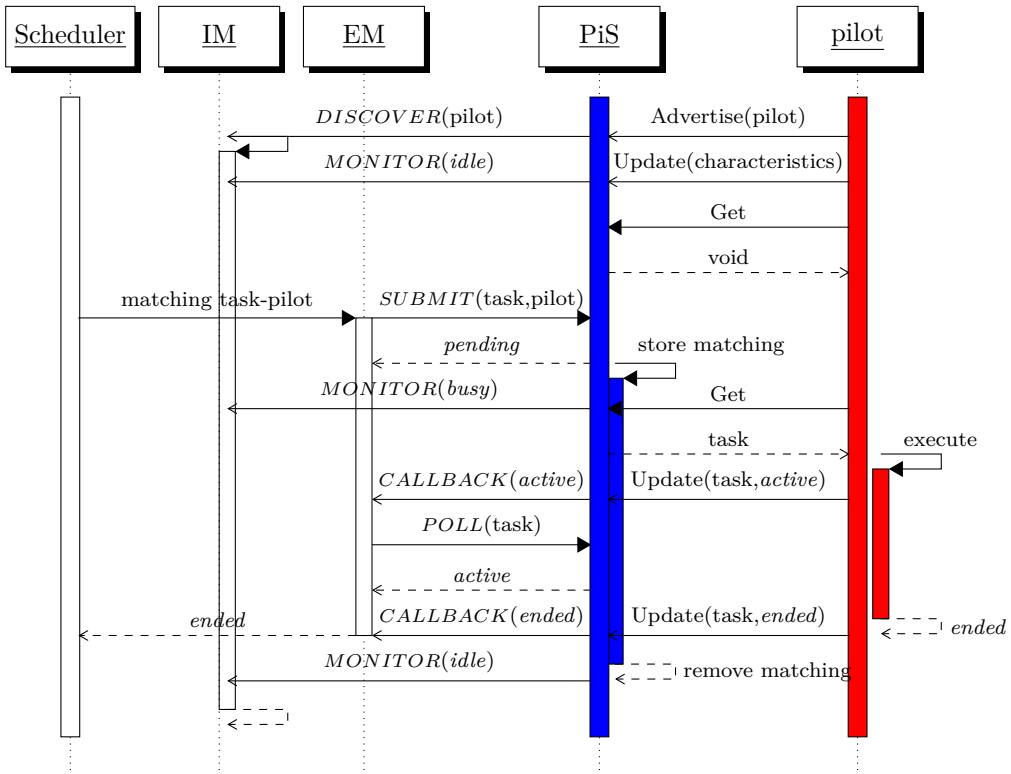


Figure 7.5: Sequence of activities performed by the actors of GWpilot to accomplish any task. They correspond to the steps 1-6 described in Subsection 7.2.2.

2. The GridWay Scheduler notices that the pilot is *idle*, so the Scheduler assigns a task from the pool to the pilot whenever its monitored characteristics fulfil the requirements of the task. Then, the GridWay EM sends the *SUBMIT* operation to the PiS through the EM adapter.

3. The PiS performs some additional checks and stores the matching. Thus, the description of the task can be only downloaded by its corresponding pilot.

4. When the pilot is *idle*, it asks the PiS for a new task. Whenever a task is fetched, the pilot immediately tries to run it and notifies the PiS.

5. The Server, through the EM adapter, immediately notifies the GridWay Core about any change in task status using the *CALLBACK* operation. Additionally, through the IM adapter, it also sends an unrequested response of the *MONITOR* operation. In doing so, the pilot becomes *busy* and the task becomes *active* for the Scheduler.

6. When the task is *ended*, the pilot notifies that either it was successfully *done* or that it *failed*. The matching is removed, and the pilot becomes *idle* again for the Scheduler.

Therefore, task states are slightly different from the aforementioned states in the pilot implementation because they must provide more information. Now, a task introduced in the Task Pool can also adopt a *pending* state, which will not change if this

task is not fetched by any pilot and it updates its status. However, the description of the task (i.e., the wrapper, streams, grid paths, as environmental variables and parameters) provided through the SUBMIT operation is only maintained in the Pool until a successful *Update Task Status* is performed by a pilot, saving memory.

However, the significance of pilot states is completely different for PiS. Pilots contained in their Pilot Pool can be *inactive* or *running*. These states stand for a pilot that is discarded by the system or accepted for processing tasks, respectively. This is so because the determination of a *busy* state in a pilot is of interest to the Scheduler, not to the PiS, which only stores the task-pilot matches from the former. The PiS only needs to know if the pilot has no tasks assigned or does not successfully update its status for a certain period of time, so it should be discarded. This deadline corresponds to the multiplication of the pulling frequency by the number of retries configured in pilots ($PI \cdot T$). The IM adapter is in charge of automatically changing the pilot state to *inactive* and updating the virtual queue tags to indicate Scheduler that no slots are available for this pilot.

Consequently, there are other situations that modify the behaviour of the GWpilot Server, such as the following:

a. If the user or the Scheduler aborts the task, the GridWay EM sends a *CANCEL* operation to the PiS, and the PiS removes the matching. Then, when the pilot tries to update the status of the task, it receives an error that is interpreted as the task being discarded.

b. When a pilot exceeds a certain number of retries for obtaining a task or reaches the queue execution limit at the remote resource, the pilot ends.

c. If the Server does not receive updates from the pilot for a time interval, it discards the pilot. Then, it informs the GridWay Core that the pilot is permanently *busy* and notifies the *failed* status if there were some task assigned to it.

Reducing overheads

In general, GridWay Core uses its manager modules (EM and IM) for sending common operations to the MADs, and then it waits for asynchronous responses. To reduce the turnaround overhead, all of them immediately respond, although some processes related to *SUBMIT* (sends a task description to certain pilot), *RECOVER* (claims for a task after a system restart) and *CANCEL* (removing a task execution request from PiS and killing it if it is being executed on a pilot) operations are performed in the background. However, GridWay Core also fills its overload with un-requested responses, so PiS uses this feature not only to later inform of submission and cancellation results but, more importantly, to improve the performance. Thus, the majority of responses for *POLL* (returns task state), *DISCOVER* (returns name identifications of active pilots) and *MONITOR* (returns pilot status) operations are originated by the PiS module adapters. It is not possible to perform the overloading technique for the other MAD operations. These responses are not arbitrarily retrieved by GridWay managers because the EM adapter immediately notifies only the task state modifications with a *CALLBACK* message, and the IM adapter caches the tag updates from *running* pilots in order to only periodically report their information changes. These un-requested *CALLBACK*, *DISCOVER* and *MONITOR* responses are the ones mentioned in the step 5.

Table 7.1: Some characteristics notified by pilots to GW PiS and subsequently published into GridWay Host Pool.

Characterisation	Tag name	Description
Identification	*PILOT_HASH_NAME*	Pilot identification code.
	LRMS_NAME	= "*jobmanager-pilot*"
Generic	*ARCH*, *OS_NAME*, *OS_VERSION*, *CPU_MODEL*, *CPU_MHZ*, *SIZE_MEM_MB*, *SIZE_DISK_MB*, *CPU_FREE*, *FREE_MEM_MB*, *FREE_DISK_MB*	Real hardware of the assigned node and generic monitoring.
GLUE-style and middleware tags	*DEFAULT_SE*, *SW_DIR*, *SCRATCH_DIR*, *DFLT_SE_FREE*, *SCRATCH_FREE*	*close-SE* and scratch directory configured. Additionally, pilot shows the vailable MB in these storages.
Virtual queue	*QUEUE_NODECOUNT*, *QUEUE_FREENODECOUNT*, *QUEUE_ACCESS*, *QUEUE_MAXTIME*, *QUEUE_MAXCPUTIME*	Number of virtual slots, available slots, accessing restrictions (VO, user distinguished name), and remaining wall and CPU time
Task	*PILOT_TASK_MEM_MB*, *PILOT_TASK_MEM_USED*, *PILOT_TASK_CPU_USED*	MB resident and the memory and CPU (%) usage by the task.
Basic network profile	*PILOT_LAG*, *PILOT_XFER_BW*, *PILOT_XFER_SE_BW*	Average lag in pilot operations and periodical bandwidth test against PiS and *close-SE*.
Caching	*CKPT_FILE_TASK*, *LAST_EXECUTABLE*, *LAST_INPUT_FILES*, *LAST_OUTPUT_FILES*	Name and MD5 of the checkpointing, staged and produced files in last execution.
User defined	*PILOT_*${*GW_USER*}*_VAR_*<*number*>	Key-Value written on pilot pipe.

With relation to pilot characteristics monitored, the information related to the host, site and user name from a discarded pilot is maintained in the Pilot Pool to be compared if another pilot is allocated in the same WN (or type in the same site) and tries to be enrolled in GWpilot. This correspondence is advised by the comparison with the identification code provided by the new pilot. However, the correspondent identifier name passed to GridWay Core through the *DISCOVER* response will always be the same, enabling system to gather performance statistics from each pilot individually.

On the other hand, the system will usually initialise one PiS on behalf of each user in order to use his certificate and to encrypt his communications. Therefore, the system can support several PiSs listening to different TCP ports and their belonging pilots are private. However, a PiS can be shared among users if their IM and EM operations pass through the same stream (the implications of this configuration are commented in Subsection 10.5).

All these processes are extensively described in this subsection because they could become an important bottleneck that usually is not analysed in other systems. In general, users perceive this overhead as an increase in the dispatching time of every task. In the case of GWpilot, the simplified design of the PiS and the streaming mechanism to communicate with GridWay Core implies that this overhead will be manageable, as will be demonstrated with the experiments from Chapter 9 to 11.

7.2.3. The GWpilot Factory (GW PiF)

The Factory is an efficient DRMAA-enabled program that follows a producer-consumer model to generate and replace pilots in a continuous flow. For this purpose, it makes the most of the producer-consumer library pattern developed in Section 4.3. To improve its performance and features, it makes use of the CLI to check the statuses of tasks and pilots. In general, a GW PiF dynamically calculates the number of necessary pilots in the system by checking if the $LRMS\_NAME =$ "*jobmanager-pilot*" sentence is set in the requirements of the tasks belonging to its particular user. Moreover, any specific characteristic notified by pilots (or even published by GIS), can also be taken into consideration for submitting pilots. This feature improves the scheduling capabilities of GWpilot in a heterogeneous environment due to it is able to distinguish the type of pilots (architecture, middleware or software installed...) running at particular sites and constraint the submission of new ones to these sites. Additionally, users do not need to worry about Provisioning because GW PiF will take account of task requirements to select suitable resources. This advanced feature and the configuration of PiF are deeply explained in Section 10.4.

7.3. Functional comparison

Two frameworks have been selected as representatives of two different approaches in the design of pilot systems: DIANE, which is perhaps the most used application-oriented framework on EGI-related infrastructures; and DIRAC as the PMS most adapted to other fields different to HEP calculations. The intention is two-fold: to compare GWpilot with other systems that propose different solutions, and to go into detail about technical issues not described in Section 2.5. These aspects are of importance if users and developers want to adapt their calculations to these frameworks and also to emphasise the advantages of GWpilot, but to properly introduce the significance of results obtained through the following Section 7.4 as well.

7.3.1. DIANE

In first place, DIANE is a small pilot system conceived to integrate the user's application into a Python framework. In addition, it is implemented to be easily configured and managed by final users. For this purpose, a short script is provided to facilitate the installation. However, although DIANE is written in Python, the mechanism also downloads binaries of external software, and even Python libraries that are not included in default OS installations. Additionally, some of them are also required by pilots (*worker agents*), and they have to be downloaded into WNs before their execution. As mentioned in Subsection 2.5.4, its most important dependence is on omniORB, because CORBA is the basis of master-worker architecture of DIANE. Moreover, the pilot submission relies on Ganga, which has its own dependencies. Thus, a complete installation requires ∼150MB, ∼70 MB out of which are external binaries (beside of basic grid middleware). These issues do not constitute a serious problem for users if they compute on CERN related operating systems, i.e. Scientific Linux (SL) 5-6, where DIANE deployment is straightforward (for example on EGI infrastructures). Additionally, an experienced user can easily compile these dependencies and modify the installation script to adapt them to other platforms.

Getting started on grid only requires an elementary configuration of Ganga, i.e.

editing some parameters in *∼/.gangarc* file. Nevertheless this implies the background use of gLite/UMD commands to remotely connect to a central WMS to perform the pilot provision. *Agent Factory* sequentially performs these operations. Additionally, there are few options to customise the Provisioning behaviour, as it can be seen in Table 7.2. Users can only perform some control on pilot scheduling if they modify the *∼/.gangarc* file for every application. These issues seriously limit the Provisioning capacity of the system, as will be depicted in Subsection 7.4.3.

Worker agents actually notify a complete set of characteristics to *Run Master*, but this feature is not completely used in the framework. For example, ranking expressions are not available. In this sense, extensions of DIANE have been implemented to take advantage of this characterisation to incorporate some generic self-scheduling algorithms into the Workload Scheduling layer. In particular, AWLB [117, 215] is able to find sub-optimal distributions of variable sized bags of tasks (BoTs) among the characterised pilots, according to a CPU and bandwidth consumption profile previously determined for the application. Nevertheless, the development of this type of approaches is restricted to advanced Python programmers. Moreover, they can expend many efforts in modifying DIANE code (the *Master*, *Scheduler*, *Factory* and *worker agents*) to accomplish the needs of certain legacy applications, which perhaps, were already implemented following a distributed computing standard (such as DRMAA or SAGA), or even following other extended specifications (such as Ganga). Additionally, task requirements can be set if *worker agents* are again modified. That is, it is needed to maintain a specific pilot by every special application. Other weakness is the impossibility of sharing pilots or statistics among running applications.

Therefore, the available Workload Scheduling for conventional users is based on FCFS. The advantages of this approach were commented in Subsection 3.2.2: maximises the usage of resources and maintains task overheads under minimum possible. For these purposes, DIANE allows the customisation of policies through $config.WorkerAgent$, $config.RunMaster$ and $input.scheduler$ variables (see Table 7.2). All of them can be dynamically set into the code, but due to their generality, only the related to the management of tasks ones are suitable to be modified during one calculation.

7.3.2. DIRAC

DIRAC is a PMS that provides an installation script following the DIANE fashion, i.e. compiled dependencies are downloaded together with the DIRAC software. However, external dependencies expend now ∼700 MB, while DIRAC software only ∼230 MB (v6r8p14 release). This implies continuous upgrades to maintain compatibility. Moreover, although the software related to LHCb can be omitted, any basic DIRAC server requires ∼40 modules (*Agents* and *Services*) running associated to a set of ten databases. Every module runs as a system daemon that continuously fills its database and generates logs which exponentially increase the disk usage. All of them must be taken into account during the installation, requiring ∼400 parameters. Many of these options have not default values and the administrator must search about their significance. The ones related to the acknowledgement of failed tasks or pilots are measured in hours or days. Some scheduling functionalities are fixed. Those options related to the comparison performed in this work are depicted in Table 7.2, and clearly demonstrate that DIRAC is only oriented to constitute a platform for multi-project production, where some skilled administrators and VO managers have the role of maximizing the throughput of long jobs.

However, the main obstacle to customise some scheduling in DIRAC is that its components are designed and optimised to efficiently accomplish workloads similar to the ones generated by the LHC experiments. As a result, its design tightly couples Task Scheduling and Provisioning to optimise these calculations, and does not tackle the possibility of setting different requirements for tasks. For example, DIRAC offers compatibility with applications based on JDL templates. That is, it provides users with commands like the gLite/UMD WMS ones. Unlike JSDL (which is used with OGSA-BES), JDL is not standardised but it is widely extended and allows the inclusion of pseudo-scripts and ClassAd expressions. Nevertheless, most of the JDL parameters related to scheduling are overlooked: users can not perform any ranking and can only constraint resources with four types of requirements (see Table 7.2). Besides JDL, DIRAC also provides developers the REST and Ganga interfaces, and its specific Python library. Nevertheless, they do not also allow the specification of more requirements.

Therefore, the scheduling is summarised in two levels: first, the system classifies tasks according to their *CPUTime* but also to other specific requirements not set by user. Subsequently, it places tasks into a queue specifically created for this classification. Posteriorly, *TaskQueueDirector* *Agent* requests any gLite/UMD WMS the information about resources that accomplish these common requirements. Then, the system filters those obtained resources according to benchmarks compiled during previous executions of pilots. Finally, one pilot per task is submitted to a concrete resource, if enough resources are listed. That is, the WMS performs the grid matchmaking, while DIRAC performs a parallel scheduling and takes the final decision about where pilots will run.

Consequences are far away from being tied to UMD middleware. First, their scheduling parameters are not really configurable by administrators. The depicted double-matching mechanism is based on specific GLUE tags and local measurements. Thus, to change the *Rank* is not recommendable to inexpert administrators because it has influence on other processes. Second, pilots can only host one task because the matchmaking is optimised for jobs whose duration approaches the maximum wall time at remote resources. One solution can be to run the *JobAgent* together with the pilot to fetch short tasks. Nevertheless, the maximum number of tasks that a pilot can accomplish is 100 (*MaxJobsInFillMode* parameter), and includes the number of retries when the corresponding queue is empty. Third, besides *TaskQueueDirector*, many other DIRAC modules participate in the scheduling process and they generate overheads. In this sense, WMS commands are relatively slow and sandboxing files in DIRAC server increases the penalty in task turnaround. These circumstances will be explained with the comparative results obtained in Subsection 7.4.3.

7.3.3. Comparison

Besides other features mentioned in Section 7.2, GWpilot differentiates among these systems by its installation (which is mainly based on OS packets and requires ∼5 MB); its configuration (which only requires ∼30 lines in two completely customisable files and some *sudo* tips); its remote compatibility and lightweight (since pilot is a short script file that only requires Python 2.4.3 as minimum release in WNs); and its management of user data (that allows staging files locally and through standard protocols). In addition, GWpilot is multiuser, allows MUPJ (opposite to DIANE) and compiles general execution statistics (opposite to DIRAC).

However, the main differences from user's and developer's point of view are that GWpilot allows them to directly run their legacy applications (written in diverse languages and following accepted standards), and fully supports the customisation of the whole scheduling at user-level, guiding both Task Scheduling and Provisioning. Thus, developers can dynamically include customised policies in these legacy applications basing them on a complete characterisation of resources without the need of modifying GWpilot code. Therefore, GWpilot is so flexible that is possible to incorporate personalised schedulers on top of the system.

Obviously, DIRAC and DIANE frameworks have advantages over GWpilot. The former has been used during years in production and can support hundred thousands of jobs and thousands of users, which has not been tested with GWpilot, neither with GridWay. Additionally it offers a complete web page that shows detailed statistics and facilitates users with different roles for their common procedures: to control site availability, to inspect tasks and pilot logs, to list statuses of tasks and cancel them, etc. In particular, the possibility of permanently banning a resource is not available in GWpilot without restart their daemons. DIANE offers the highest performance in terms of overhead in Task Scheduling when the default FCFS was used. An evidence of this affirmation is that the option *PULL_REQUEST_DELAY* must be set to avoid the overload of *Master*.

7.4. Reproducible comparison with other pilot systems

The aim of the experiments is to demonstrate the viability and suitability of GWpilot, as well as its performance as an improvement achieved in comparison with other pilot systems, when common scientific applications are executed on resources belonging to large production DCIs. For this purpose, GWpilot must be tested on a real infrastructure and measured with applications that create non-ideal conditions for a computational distributed environment, i.e., filling the system with a high volume of variable-duration short tasks in a continuous flow, which must be dispatched individually. This approach is of enormous importance because centralised pilot systems usually validate their performance only with long tasks. This methodology is well founded for the specific calculations for which these systems were initially implemented, i.e. the LHC production. However many user applications are composed of short tasks that should profit from their extensive distribution.

The objective is not to expose the advantages of certain complex scheduling algorithms that GWpilot can incorporate but to show how the default GWpilot functionalities result in a valuable improvement over other systems.

Thus, the behaviour of two representative frameworks (DIANE and DIRAC) is compared with GWpilot. To better analyse the results obtained, and because DIANE only can run in this mode, one-user and one-task pilot jobs over a unique grid VO will be utilised. Additionally, general issues on their configuration and adaptation are commented, not only to be contrasted with GWpilot capabilities, but also to establish the same scheduling requirements to perform equivalent tests among them. Thus, the basic performance of GWpilot is properly measured with respect to some existing approaches, and therefore an important performance gain is expected for any other feature of GWpilot to be used.

Table 7.2: Scheduling policies with similar significance for the pilot systems compared in this work. Values set to accomplish the experiments are shown.

	GWpilot	**DIANE**	**DIRAC**
Provisioning new pilots:			
Max. amount and overload	PiS option ($gwd.conf$), PiF params. (manually)	*Agent Factory* or *diane-submitter* ($diane-worker-number$) param. (manually)	Same as number of tasks and adds: $TaskQueueDirector/extraPilots(=2)$
Max. suspension in LRMS or GS	PiS option ($gwd.conf$), PiF param. (manually)	*Agent Factory* or *diane-submitter* ($pending-timeout$) param. (manually)	*ExpireTime* is not set in the pilot JDL, then remote WMS waits its default timeout (typically 1 day)
Submission limits	$DISPATCH\_CHUNK(=100)$ ($sched.conf$)	It is secuential, but Agent Factory and diane-submitter limit with ($diane-max-pending$) param. (manually)	$TaskQueueDirector/pilotsPerIteration(=100)$
Ranking resources	PiS option ($gwd.conf$), PiF param. (manually), or collected from any $RANK$ in tasks (dynamic)	*Rank* option ($\sim$ /.*gangarc*), or Agent Factory ($square-fitness$) param. (manually set, but it is a fixed policy, based on completion rate: $[(running+completed)/total]$)	$TaskQueueDirector/glite/Rank$ (Expression based in GLUE descriptions of CEs obtained from top-BDII)
Constraining Resources	PiS option ($gwd.conf$), PiF param. (manually), or collected from any $REQUIREMENT$ in tasks (dynamic)	*Requirement* option ($\sim$ /.*gangarc*), or *Agent Factory* or *diane-submitter* param. (manually, but only allows list of CEs, i.e. round-robin)	Fixed in code and based on *Rank* expression and HepSpec06 of CEs also obtained from top-BDII. Additionally, only 4 requirements are also collected from task JDLs to generate constraints: $CPUTime$, $Site$, $BannedSites$, and $Platform$
External broker	—	$glite\_wms.conf$	$TaskQueueDirector/glite/ResourceBrokers$
Pilot behaviour:			
Pulling new task	Inmediately, and then $PI(=30s)$	$config.WorkerAgent.$ $.PULL\_REQUEST\_DELAY(=0.2s)$	$JobAgent/SubmissionDelay(=10s)$, and then $120s$
An *idle* pilot ends	$T \cdot PI(=20 \cdot 30s)$	One attempt every: $config.WorkerAgent.HEARTBEAT\_DELAY(=10s)$ Pilot ends if an attempt last more than: $config.WorkerAgent.$ $.HEARTBEAT\_TIMEOUT(=30s)$	$JobAgent/StopAfterFailedMatches(=10)$ failed attempts, or when the number of completed tasks and failed attempts reachs to: $TaskQueueDirector/$ $glite/MaxJobsInFillMode(=100)$

(Continue in next page.)

(Table 7.2 continued.)

Policies to discard pilots:			
Outage	$T \cdot PI(= 20 \cdot 30s)$	$config.RunMaster.$ $.LOST\_WORKER\_TIMEOUT(= 60s)$	$PilotStatusAgent/PilotStalledDays(= 3d)$
Idles	$T \cdot PI(= 20 \cdot 30s)$	$config.RunMaster.$ $IDLE\_WORKER\_TIMEOUT(= 600s)$	—
Task failure	Immediately	$input.scheduler.policy.REMOVE\_$ $\_FAILED\_WORKER\_ATTEMPTS(= 1)$	$JobAgent/StopOnApplicationFailure(= true)$ (on first task failure, pilot ends)
Policies to manage tasks:			
Execution attepms	$RESCHEDULE\_ON\_FAILURE$ $(= no)$, $NUMBER\_OF\_RETRIES$ $(= 3)$ in tasks (dynamic)	$input.scheduler.policy.$ $.FAILED\_TASK\_MAX\_ASSIGN(= 3)$ $.LOST\_TASK\_MAX\_ASSIGN(= 3)$	Reschedule fixed in 3 retries
Avoid ending pilots	$REQUIREMENT =$ "$QUEUE\_MAXTIME > (30m)$" in tasks (dynamic)	$input.scheduler.policy.$ $.WORKER\_TIME\_LIMIT(= 0s)$ (max. time in execution)	JobAgent compares $CPUTime$ requirement set in task JDL and the remaining time notified by pilot before matchig. B
Matching time	$SCHEDULING\_INTERVAL$ $(= 10s)$ $(gwd.conf)$	Inmediately	$TaskQueueDirector/ListMatchDelay(= 10s)$
Avoid overloaded or failing pilots	$SUSPENSION\_TIMEOUT$ $(= 61s)$ in tasks (dynamic)	$config.RunMaster.$ $.LOST\_WORKER\_TIMEOUT(= 60s)$	$StalledJobAgent/StalledTimeHours(= 6h)$ $StalledJobAgent/FaledTimeHours(= 2h)$

7.4.1. Test bed setup

The *fusion* VO has been used for the calculations because it offers a large, heterogeneous and overloaded number of resources shared with other highly demanding VOs. When the tests were performed, the *fusion* VO counted on more than 40 computing elements (23 sites), and up to 29,464 slots. Of course, few of them are actually available, but every framework should obtain 1,000 slots. This value is used as the minimum amount of pilots that every system will request during the experiments.

To perform accurate comparisons among frameworks, three identical virtual machines (4-cores, 24 GB RAM; SL 6.3; UMD UI 2.0.2-1) are configured with DIRAC (v6r8p14 release), DIANE (2.4) and GWpilot (on GridWay 5.14), running on a dual Xeon X5560 (16 cores, 2.8 GHz). Every instance stores their logs, user data and databases on virtual disks created on RAM. This is done to avoid the interference among pilot systems on servers'performance, because only one framework is started in every virtual machine booted, and the intensive I/O operations are isolated in every reserved memory. Additionally, three different user certificates are used during the experiments. This is to assure a fair-share between tests at remote queues, a key point that is indispensable to achieve accurate results on an overloaded infrastructure such as the *fusion* VO in EGI.

To ensure the accuracy of the results obtained, tests are run in parallel and configuration options are set as similar as possible. Table 7.2 shows these configuration equivalencies which are also taken as a model for later experiments. In this sense GWpilot parameters are adapted to be as the default policies used in DIANE. However, the default options are maintained for DIRAC with the exception of the activation of filling mode, because no comparison would be possible without it. Therefore, this is the configuration found by any user at central DIRAC servers.

To eliminate the impact of Provisioning policies, resource ranking and filtering capabilities are disabled on GWpilot for these experiments. PiFs (GW PiF and *Agent Factory*) have been restricted to create a maximum of 1,000 *running* pilots. The maximum suspension timeout, before cancelling a pilot job whenever it is queued at the remote LRMS for a long time, has been set to 30 minutes. This value is considered reasonable according to the duration of tests (i.e. 3-10 hours). Additionally, it provides a level playing field with DIANE's Provisioning mechanism, even though GWpilot could work better with lower timeouts [135, 227]. However, banning feature is enabled (set to 1 hour) in GWpilot. This is necessary to avoid sending pilots again to the same failing resources (the suspension timeout is considered as a failure in the experiments). Thus, to improve in a similar way the Provisioning in DIANE executions, the *square-fit* option is passed to *Agent Factory*. This implies that DIANE will guide the Provisioning process for some tasks. Moreover, DIRAC will also perform by default the profiling of every site and will use statistics to select resources. Therefore, GWpilot is *a priori* put at a disadvantage in the Provisioning phase. Finally, caching files is also disabled in GWpilot, while DIANE and DIRAC perform staging operations as usual.

However, other GWpilot parameters must also be set to perform any calculation, although they effectively limit the usage of the infrastructure. They were configured to allow the submission of a maximum of 100 pilots per site (which is defined with the *MAX_RUNNING_RESOURCE* statement). Additionally, GWpilot is allowed to schedule a maximum of 100 tasks (to pilots) and pilots (to sites) every 10 seconds. These values are set for similarity with DIRAC, although they limit the capacity of GWpilot to access resources with respect to the other frameworks.

Table 7.3: Time complexity of long multiplication and real time measurements.

<Number of task> mod 10	$\Theta(n^2)$	Xeon X5365 3 GHz (Launch date: 2007)	Xeon E5620 2.4 GHz (Launch date: 2012)
Very short tasks: $< lower\ limit >= 3$ $(n = 30,000 \cdots 75,000)$			
0	$\Theta((3 \cdot 10^4)^2)$	66.97 s	84.50 s
1	$\Theta((3.5 \cdot 10^4)^2) = (1.1\overline{6})^2 \cdot \Theta((3 \cdot 10^4)^2)$	91.18 s	115.03 s
6	$\Theta((6 \cdot 10^4)^2) = 2^2 \cdot \Theta((3 \cdot 10^4)^2)$	269.64 s	338.31 s
8	$\Theta((7 \cdot 10^4)^2) = (2.\overline{3})^2 \cdot \Theta((3 \cdot 10^4)^2)$	369.03 s	459.81 s
9	$\Theta((7.5 \cdot 10^4)^2) = (2.5)^2 \cdot \Theta((3 \cdot 10^4)^2)$	419.43 s	529.27 s
Short tasks: $< lower\ limit >= 9$ $(n = 90,000 \cdots 135,000)$			
0	$\Theta((9 \cdot 10^4)^2)$	605.21 s	761.04 s
1	$\Theta((9.5 \cdot 10^4)^2) = (3.1\overline{6})^2 \cdot \Theta((3 \cdot 10^4)^2)$	672.86 s	851.62 s
6	$\Theta((12 \cdot 10^4)^2) = 4^2 \cdot \Theta((3 \cdot 10^4)^2)$	1,074.32 s	1,354.96 s
8	$\Theta((13 \cdot 10^4)^2) = (4.\overline{3})^2 \cdot \Theta((3 \cdot 10^4)^2)$	1,262.12 s	1,587.38 s
9	$\Theta((13.5 \cdot 10^4)^2) = (4.5)^2 \cdot \Theta((3 \cdot 10^4)^2)$	1,360.91 s	1,713.19 s

Finally, the last two parameters in the GWpilot configuration determine the pilot turnaround overhead and the pilot discarding rate of the pilot system, respectively: the pulling interval (PI) for GWpilot, which is established to 30 seconds, and the maximum retries number (T), which is set to 20. These values are selected to be similar to the discarding timeout for DIANE's idle *worker agents* ($IDLE\_WORKER\_TIMEOUT$).

The shorter duration of tasks implies a higher impact of overheads on the computation performance and on the load supported by systems. In addition, most users do not want to deal with grouping their tasks. Developers are also aware of how the failure rate increases when big BoTs are used, and would rather massively distribute minimal tasks to reduce the final makespan. For these reasons the comparison among pilot systems is made with short tasks. Additionally, the intention is to simulate the typical behaviour pattern of a user that submits a daily calculation, for example, carried in background during the night. Therefore, tests with a maximum wall time of 5-10 hours are selected for these experiments.

7.4.2. Simple calculation

For the experiments, long (or standard) multiplication is selected to be easily measurable and reproducible. The time complexity of multiplying two *n-digit* numbers using long multiplication is $\Theta(n^2)$. Thus, task duration can be easily controlled modifying n as shown in Table 7.3. To obtain these data, long multiplication was implemented in C language and compiled with *gcc* 4.4, resulting in a binary of 13KB. The executable requires the *n-digit* as a parameter to randomly generate two numbers of that size. Then, it multiplies both of them, and stores the result in a file of $\sim 6 \cdot n$ Bytes.

To generate an effect similar to a real user calculation (but also controlled for further analysis), tasks are submitted with different n values along the tests following this pattern:

$$< lower\ limit > \cdot 10^4 + (5 \cdot 10^3 \cdot (< number\ of\ task > mod\ 10))$$

Thus, setting $< lower\ limit >$ to 3 assures that the duration of task ranges between $\sim$ 67 s and $\sim$ 420 s on the first machine mentioned in Table 7.3, which is

taken as a lower reference machine of the infrastructure due to their manufacture date. However, significantly lower execution times are not expected in this case due to long multiplication only depends on the velocity of the processor.

Perhaps, processing tasks that last less than five minutes has not much sense in a distributed environment (with the exception to of real-time applications). The sole purpose to carry on this type of calculation is to perform a stress test of the pilot systems. For this reason, a second test that contains an input set that guarantees task duration above 10 minutes is also performed, i.e. setting $< lower\ limit >$ to 9.

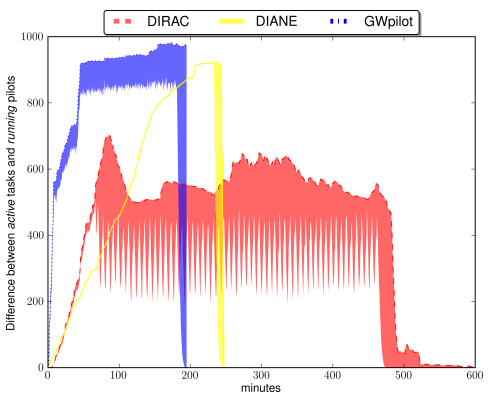
The number of tasks of both tests can be determined according to the lower reference machine. $3 \cdot 10^4$ tasks would last approximately 1 hour and 51 minutes on 1,000 X5365 cores, and would generate 8.8 GB as output if a $< lower\ limit >= 3$ is selected, while the same volume of tasks would last 8 hours and generate 18.86 GB when a $< lower\ limit >= 9$ is set. However, these calculations will last much more due to the real availability of resources in grid.

A script that creates JDLs and makes use of commands was implemented to port long multiplication to DIRAC. This script is as simple as any conventional user will implement. It sequentially submits and checks the statuses of tasks. If ended tasks are detected, outputs are checked before to submit new tasks. Else the script waits 60 s. However, unlike the other approaches that directly store the outputs in a local user directory, DIRAC is based on the sandboxing of these outputs to be downloaded later by the user. It is noteworthy to mention that DIRAC provides commands for searching for text in the output files without having to download them. Nevertheless, they can last minutes due to the volume of tasks managed in this experiment. These delays avoid any comparison with the other pilot systems and they are not used. Additionally, every task should be deleted after downloading its outputs and checking if they are correct. Every operation requires 1.2-1.5 s despite of the script is launched on the same machine that runs DIRAC to remove network overhead. This behaviour is unfeasible because only checking the statuses of 1,000 tasks would last more than 2 minutes. For this reason, every command launched will manage up to 100 tasks together.

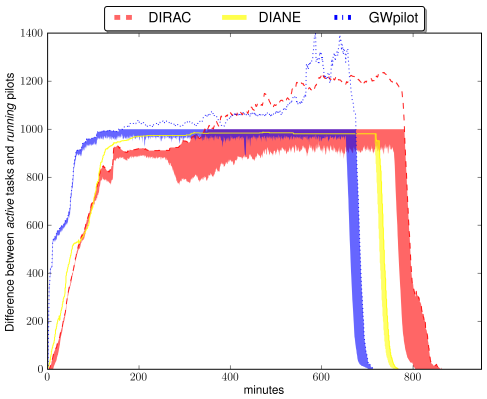
To make a more proper comparison and to measure the performance of GWpilot CLI, a similar script was implemented to generate templates and manage the long multiplication tasks through GWpilot. However, with DIANE, there is no other possibility than to wrap long multiplication into DIANE libraries, and so it was performed. Naturally, both implementations will check the outputs of every task to ensure the completeness of results. The maximum number of managed tasks is limited to 1,000 in the three implementations. With respect to the basic policies for managing tasks in GWpilot, the same approach for Provisioning is followed: neither ranking, nor requirements were dynamically set on tasks by the application, with exception of the necessary ones to avoid ended or failing pilots (see Table 7.2). The task suspension timeout has to be set slightly above the pilot pulling interval in order to quickly remove tasks from pilots that could be failing.

7.4.3. Results

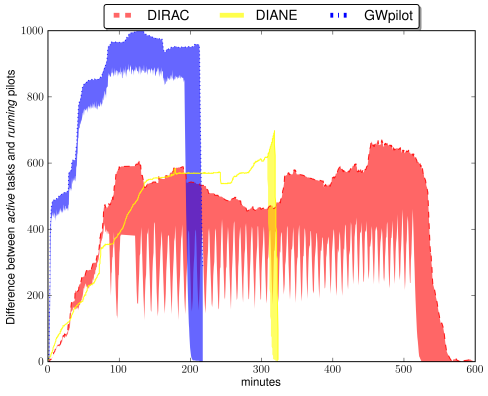
Both short ($< lower\ limit >= 3$) and long ($< lower\ limit >= 9$) tests were carried out three times to assure accurateness. Makespan obtained with every pilot system is listed in Table 7.4, which must be provided as the principal speedup measurement from a user point of view. However, to allow a proper comparison among systems and



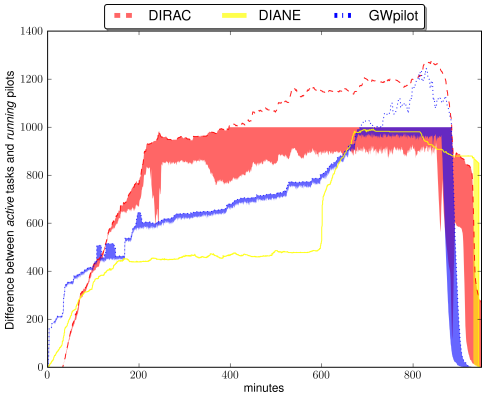
(a) $n = 30,000 \cdots 75,000$



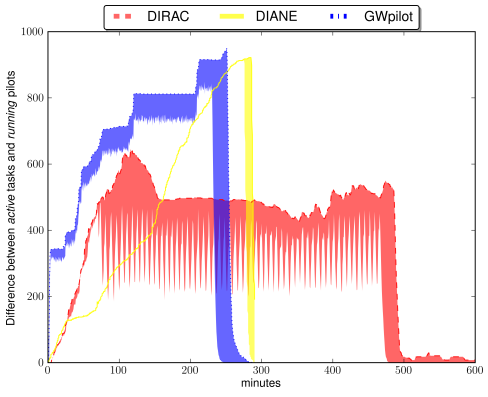
(d) $n = 90,000 \cdots 135,000$



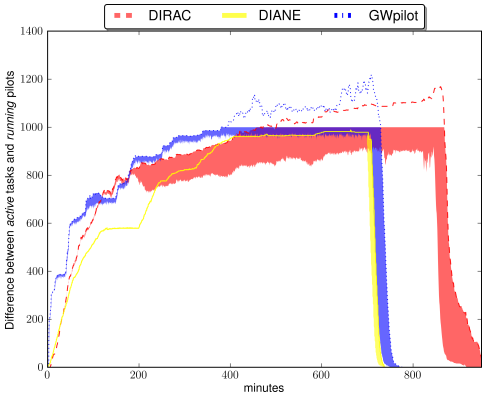
(b) $n = 30,000 \cdots 75,000$



(e) $n = 90,000 \cdots 135,000$



(c) $n = 30,000 \cdots 75,000$



(f) $n = 90,000 \cdots 135,000$

Figure 7.6: Difference between the number of *running* pilots and *active* tasks (blue represents GWpilot, red represents DIRAC, and yellow represents DIANE). There is an upper limit of 1,000 managed tasks per application run on every pilot system. Left (right) column cases a, b and c (d, e and f) corresponds to three different tests with $n = 30,000 \cdots 75,000$ ($n = 90,000 \cdots 135,000$).

Table 7.4: Results obtained in the comparative experiment. Values of pilots submitted between parenthesis correspond to the number of pilots guided by DIANE *Agent Factory* to a specific site. Values for *mean up* and *filling rate* are measured before the last 1,000 tasks were remaining in the systems due to the end of calculation is not representative of the usual utilisation of resources.

Test	**System**	**Makespan**	**Tasks failed**	**Pilots**			
				submitted	enrolled	mean up	filling rate
Short tests (very short tasks): $< lower\ limit >= 3$ $(n = 30,000 \cdots 75,000)$							
	DIANE	4 h 06 m 49s	466	2,563	1,381	525.08	99.76
(a)	DIRAC	7 h 56 m 54s	4	2,165	1,874	518.73	63.50
	GWpilot	3 h 13 m 32s	82	2,490	1,011	797.72	86.85
	DIANE	5 h 22 m 26s	1991	3,431	2,797	428.90	99.80
(b)	DIRAC	9 h 03 m 01s	8	2,726	1,751	460.91	62.98
	GWpilot	3 h 27 m 01s	135	2,996	1,054	785.62	86.99
	DIANE	4 h 50 m 42s	608	3,323	1,570	421.98	99.75
(c)	DIRAC	8 h 21 m 56s	6	2,049	1,297	461.74	68.50
	GWpilot	4 h 06 m 42s	62	4,655	954	648.23	87.80
Long tests (short tasks) $< lower\ limit >= 9$ $(n = 90,000 \cdots 135,000)$							
	DIANE	13 h 16 m 02s	522	1,989(932)	1,531	893.99	98.63
(d)	DIRAC	13 h 24 m 00s	486	4,226	3,454	968.97	87.29
	GWpilot	11 h 33 m 32s	183	9,032	3,687	1,001.63	94.49
	DIANE	15 h 29 m 36s	3,140	6,435(2330)	4,137	588.45	99.93
(e)	DIRAC	16 h 06 m 03s	650	12,888	3,095	953.03	85.43
	GWpilot	15 h 08 m 36s	589	16,369	2,213	670.16	95.18
	DIANE	12 h 16 m 17s	2,322	3,919(1393)	3,295	774.32	99.62
(f)	DIRAC	15 h 19 m 22s	822	3,135	2,537	894.35	88.58
	GWpilot	12 h 34 m 49s	151	9,444	2,645	880.0	94.61

against other approaches, other performance metrics used in previous works [204, 258, 201, 195] are considered. Thus, in addition to the final makespan parameter, Table 7.4 compiles the number of pilots submitted and effectively enrolled as a measurement of the efficiency in Provisioning. Additionally, the average number of pilots actually running and the filling rate of these pilots with tasks are provided. These values are measured before the last 1,000 tasks were remaining in the systems because the end of calculation is not representative of the usual utilisation of resources in any experiment. The number of failed tasks is also added.

To complement this information and illustrate the evolution of the tests, Figures 7.6-(a, b, c, d, e, f), show the difference between the number of *running* pilots and the number of *active* tasks in every framework. These figures have been obtained by processing DIANE (*master.j*), DIRAC (*Matcher* and (*JobStateUpdate*), and GWpilot logs.

First observable issue in the figures is the improved capacity of GWpilot for Provisioning. It is able to obtain and retain the requested resources one or two hours before than the other systems. Performance achieved by GridWay in early-binding scheduling is known, and results obtained in GWpilot tests should be better than the ones using gLite/UMD WMS, according to previous works [135, 227] or the analysis performed in Chapter 6. However, this deserves a deeper explanation because most of the scheduling advantages of GWpilot have been disabled in this experiment and the overload generated by pilots is added. This capability is consequence of the modular design of GWpilot. MAD in charge of controlling job execution in CREAM sites conceptually

follows a similar implementation to GW PiS: both manage the continuous data flow from and to system core and remote sites. Additionally, it uses minimal middleware libraries, unlike middleware commands. However, the Scheduler is always performing the match-making among pilots and resources, and among tasks and pilots. For this purpose the system maintains the Scheduler aware of the data flow coming from GW PiS, CREAM and IM MADs. With this information Scheduler performs its match-making algorithm over a limited number of pilots and tasks ($DISPATCH\_CHUNK$), once per interval ($SCHEDULING\_INTERVAL$). Therefore, due to its modular architecture, CREAM driver and PiS can potentially manage thousands of pilots and tasks more quickly and concurrently than DIRAC or DIANE solutions. In fact, it is mainly limited by the computational complexity of the algorithm implemented in the Scheduler, as will be explained (and improved) in Chapter 9.

Unlike GWpilot, the other frameworks use gLite commands to submit pilots (or even to perform match-making requests in the case of $TaskQueueDirector$). Every contact with the WMS spends above 3 s. DIRAC implements such capability by a multi-threaded engine and is less exposed to these overheads. Nevertheless, it offers a similar performance to DIANE if the number of failed tasks is observed. Every failed task triggers the discarding of its hosting pilot (see Table 7.2) in DIANE. Submission performed by *Agent Factory* is completely sequential, and subsequently the maximum number of slots provisioned is limited to approximately 1,000 per hour for DIANE. On the other hand, GWpilot is able to submit this volume of pilots in 100 seconds, according to the configuration selected in this work. Thus, GWpilot can immediately start the replacement of failing or suspended pilots. However, this feature can not be enough in certain situations as the one presented in test (e), where the infrastructure is overloaded. This circumstance has already observed for early-binding in Chapters 4 and 6: in some cases, allowing large suspension times at remote queues can improve the Provisioning (see DIRAC behaviour in Figure 7.6 -(e)).

In view of the results, GWpilot is clearly more adequate to accomplish short duration experiments, as the ones described in Figures 7.6-(a, b, c). In experiments longer than 15 hours, the benefit is slightly lower. However, when many pilots end due to reaching the maximum wall time at remote queues (usually 24 hours running), the improved Provisioning capacity of GWpilot will maintain the number of resources appropriated unlike other solutions.

Despite of the drawback, it seems that DIANE progressively trims the advantage that GWpilot holds in some tests (f, a, c), although not in others (d). That is because its task completion rate is higher than the one achieved by GWpilot when it reaches a similar number of pilots (a, c) and because *Agent Factory* could be appropriating more powerful resources. It is noteworthy to mention again that DIANE performs such a type of guided Provisioning (see Table 7.4) only on long tests (with $< lower\ limit >= 9$). That is so because *Agent Factory* needs time to compile enough data about *running* pilots in order to start submitting new ones to certain resources. DIRAC is always evaluating performance of pilots to select suitable resources. However, these features do not seem to be an effective advantage over GWpilot through the tests. Other reason could be the existence of overheads, which will be discussed in the following paragraphs.

The effective resource utilisation is a clear indication of how overheads decrease overall performance. This is usually measured using the area between *running* pilots and *active* tasks graphs [258]. DIANE execution is plotted as a line, with the exception of the end of computation when many pilots are still up, but there are no more

tasks to execute. That is so because DIANE reaches average filling rates of about 99 %, i.e. it performs the highest pilot utilisation, as expected: with DIANE, any ended task is immediately detected by the user application because the application itself is embedded in DIANE. Thus, a new task that substitutes the previous one is immediately generated, and subsequently dispatched. This process takes less than 0.4 s for every task because it has no scheduling overhead associated. The FCFS approach does not require expending time in solving match-making algorithms.

The jagged shape of the displayed *active* tasks line in GWpilot and DIRAC executions is due to their arbitrary termination. It is justified by the alternation of light tasks with heavy ones proposed as test input. Although this behaviour could be attenuated by decreasing the pulling (PI) and the scheduling interval, or increasing the dispatching chunk, it is also determined by the implementation of the script that manages the execution of the application. This script is not able to process and supply new tasks to GWpilot when their duration is very short (these overheads will be properly described in Subsection 11.2.2). On the other hand, match-making complexity is so simple in this experiment that it takes the Scheduler less than one second to solve it. Thus, the overhead originated by Scheduler is an issue to be analysed through other type of experiments (see Chapters 9 and 11), where ranking is extensively used. However, other overheads have influence on the task turnaround time in this experiment and reduce the task completion rate. One of them is file staging, but another one is the number of failed tasks that trigger the banning of pilots by a period of time (see Table 7.4). However, overheads are actually appreciable with task duration below five minutes. Results obtained in other cases seem to be comparable with those of a FRFS scheduling approach [258], or even in [195], as shown in Figures 7.6-(d, e, f). This is indicative of the design and implementation feasibility of GWpilot.

Other added overheads are hindering fill the pilots in DIRAC executions. In short tests (a, b, c), the slowness of CLI commands dramatically decreases the number of tasks submitted. That is due to the great number of DIRAC modules that are working together to process the amount of requests coming from the application. These processes maintain the 4 CPU cores of virtual machine above 60 % of usage. Consequently, commands take up to 20 seconds to return the results. This is the reason for the pronounced jagged shape of running tasks in these tests. The drawback is clearly attenuated in long tests (d, e, f), however scheduling and pilot overheads still prevent reaching filling rates similar to the other approaches. Explanation is simple: although *Matcher* is able to dispatch most of the tasks in milliseconds [204], tasks queues are empty many times. Then, many pilots must wait 120s to try again getting a task. To classify tasks in queues is not an immediate mechanism, as well as the process in which the tasks are considered completed. Thus, there is a delay between accepting a task and dispatching it; and there is another delay between the task being done and its output being available for download. In the case of long tests (d, e, f), commands do not have influence on the filling rate because the script always maintain 1,000 tasks in the DIRAC system without problems. Nevertheless, the script can not detect finished tasks although their hosting pilots had completed the work. Additionally, the sandboxing mechanism delays the processing of ended tasks. Consequently, many tasks in long experiments are in unproductive states.

Therefore, the underutilisation of pilots and the delay on obtaining results is a problem for individual users that plan to use DIRAC to execute applications composed by short tasks. However, this issue is not a drawback for the managers of large collaborative projects where many clients use PMS at the same time, because pi-

lots can always be filled with other user's tasks if the MUPJ capability is enabled. Thus, DIRAC will maintain the appropriate throughput and resource utilisation to accomplish its associated projects.

7.5. Conclusions

In this chapter, a general-purpose pilot framework created to accomplish requirements listed in Subsection 3.2 has been presented: the GWpilot framework. Its complete design, internal operation and features have been detailed. Furthermore, the framework was functionally compared with two representative and customary pilot systems. In this sense, a first step has been performed by comparing the basic performance of GWpilot with these two systems. With both functional and experimental comparison, these GWpilot functionalities have been demonstrated:

- Achievement of better performance in Provisioning.
- Accomplishment of the basic Task Scheduling of short tasks with minimum overheads.
- Support of a friendly CLI without delays.
- Easy deployment without requiring modifying its code.
- Complaining with grid security and protocols.

In particular, the performance achievements gains significance when the experiments were performed without enabling the advanced scheduling features of GWpilot. However, to fully support Multilevel Scheduling the system must allow users to characterise resources as well as to build Workload Scheduling algorithms that influence on Provisioning. For this purpose, an advanced Task Scheduling are provided. Moreover, the system supports legacy applications and observes fair-share rules in multiuser environments. It is also desirable that the framework can profit from resources belonging to several grid and cloud federations. All these aspects have been considered in the design of the framework, but require a wide explanation and demonstration, which are performed through the following chapters of this thesis.

Chapter 8

Simple Provisioning for Legacy Applications

8.1. Introduction

GWpilot is designed to profit from the numerous advantages of GridWay; among them, it incorporates interfaces for some of the few established standards in Distributed Computing to implement applications [47], such as DRMAA, JSDL and OGSA-BES. Additionally, the user-friendly CLI from GridWay is available, and therefore, any user can discretionally submit tasks to pilots through the *gwsubmit* command, as has been shown through the previous experiments. It is important to note this aspect because not only the applications already running on GridWay, but also any application that observes those standards can profit from GWpilot. This way, the impact of the solution proposed in this work is even higher.

The aim now is to show how any legacy application can be straightforwardly adapted to GWpilot and consequently will make use of grid, but also of cloud resources if the GWcloud drivers described in Chapter 5 are configured. The intention is to focus on this last feature, because the advantages of combining GWpilot and GWcloud have not been achieved before by other systems and they constitute a step forward in the use of IaaS clouds for accomplishing HTC calculations. Therefore, although the compatibility with legacy applications is a distinguishing advance by itself, other features are demonstrated such as decentralisation, middleware independence, easy configuration, and lower overheads that allow local installations, even on the PC of the user. Moreover, the capacity for accomplishing short tasks on resources geographically distributed and provisioned on-demand from cloud federations is a fundamental achievement because the dynamic cloud brokering is very limited in practice, as was shown in Chapter 2. The compatibility with grid standards or protocols is another feature that increases the adaptability among infrastructures.

Legacy applications are benefited from these features almost without performing modifications in their codes as few parameters have to be set in the configuration of GWpilot. To achieve this goal, the application scope and the Task Scheduling and Provisioning must be clearly differenced from the user's and developer's point of view. Legacy applications rely on the Task Scheduling supported by the framework to straightforwardly distribute the *computation* (see Subsection 2.3.6). However, users

should be aware of the used resources that are being provisioned, and can be constrained for their specific needs, especially when a special virtual environment is required. These issues are tackled through the following sections.

8.2. Straightforward adaptation of legacy applications

To understand the mechanism, any developer or user should have in mind that pilots actually execute the same wrapper as that used by GridWay for the regular submission of grid jobs. This wrapper downloads the necessary files to execute the user task, manage its execution, and subsequently store the outputs in the GridWay host or somewhere in the grid. Therefore, the significance of any file or variable declaration in the task description is maintained with pilots. This implies that the file staging is also performed through the standard GridFTP or the GASS protocols. Thus, the only condition to adapt any application to GWpilot is to simply add $LRMS\_NAME =$ "*jobmanager-pilot*" in the requirement statement of every task because it indicates to the Scheduler that these tasks must be submitted to a pilot. This statement distinguish pilots from other type of resources in the GridWay host pool, but it is one out of many tags that pilots publish that can be used to introduce specialised Workload Scheduling policies.

All of these aspects are shown in the examples for JSDL, DRMAA and GridWay templates in Figures 8.1-(a, b, c), respectively. The three codifications declare the same task. They specify the location of a text file, which has to be taken as the input of a *tail* command, and also where its output must be stored. However, there are three additional statements that are important to correctly understand for the adaptation of applications to the system. The *REQUIREMENT* statement has to be used as previously commented to constraint the task execution to pilots, but obviously, it can be used to restrict the execution to some of them. For example, it is desirable that tasks will not fall into pilots near to end. This aspect is important for grid executions due to the wall time limitations of batch queues, even for cloud when providers follow a pay-per-use policy. Thus, as the *QUEUE_MAX* tag shows the remaining wall time of every pilot, a minimum value can be set as an additional requirement. Moreover, *RANK* and *SUSPENSION_TIMEOUT* are newly introduced. They have been deliberately included in the examples shown in Figures 8.1-(a, b, c) because they are repeatedly used in the experiments performed in this thesis, although any other statement allowed in templates can be declared by similar translations in JDSL and DRMAA. It is noteworthy that these scheduling features are only provided by GridWay, but their declaration is allowed as extensions of the DRMAA and JSDL specifications. Therefore, the compatibility of the applications with other systems is guaranteed.

Therefore, including any constraint or ranking policies in the description of tasks is indispensable to preserve any Workload Scheduling already implemented in the application. Furthermore, it is useful to build a new specialised scheduling for the adapted application. As these pilot tags are visible through the *gwhost* command, developers can easily select the adequate ones for this purpose. For example, ranking pilots is very simple because it only requires setting a numerical tag in the rank parameter, that is, writing in the task template the sentence $RANK =< tag\_name >$ or their equivalent statements in JSDL or DRMAA. This is the case of the examples of

Figures 8.1-(a, b, c), where the ranking relies on the CPU speed (i.e., the *CPU_MHZ* tag).

Alternately, the *SUSPENSION_TIMEOUT* is the maximum waiting time at the remote batch system that GridWay allows before cancelling a regular job. However, the suspension timeout acquires a new significance when pilots are used. No batch queues are in pilots (although GridWay manages them as queues), and tasks are immediately executed. Thus, this waiting time is the one needed by a pilot to fetch a task and update its state, which is related to pilot pulling interval (PI). For this reason, the configuration of the pulling interval is very important: it determines the overhead introduced in the Task Scheduling, which allows developers to estimate the better size of BoTs. Furthermore, if the banning feature is enabled, a pilot that repeatedly ignores tasks beyond the timeout set will be ignored for future scheduling. In general, users must set the *SUSPENSION_TIMEOUT* slightly higher than the pulling interval established to effectively discard failing pilots.

Obviously, the tag mechanism can be used for implementing complex scheduling algorithms to coordinate both Workload Scheduling and Provisioning. However, it is out of scope of this chapter to thoroughly explain the possibilities that GWpilot can offer to build those schedulers. The intention is to present how the basic features can allow tuning the applications straightforwardly adapted to the framework. It is worth mentioning anyway that in addition to the generic characteristics notified by pilots, such as bandwidth, CPU type or usage, users can customise new ones. These tags can be also obtained on-demand through the *gwhost* command, and developers can use them to implement dynamic schedulers that select the adequate pilots for their tasks, and even their own pilot factories. The possibilities and the tools offered for these purposes will be deeply explained in Chapters 10 and 11.

8.3. Customised Provisioning

8.3.1. Provisioning in grid federations

The basic Provisioning capabilities of GWpilot on grids were extensively demonstrated in the previous Chapter 7. Thus, the support of legacy applications can be easily inferred from them. Moreover, experiments detailed through the following Chapters 9 and 10 are performed on grid making use of several legacy applications. However, the configuration options that allow correctly profiting from these features must be clarified to any user or developer, because there are concepts that sound similar to the ones used in Task Scheduling, which meaning changes for Provisioning.

First, system counts on several drivers to obtain resources from several types of infrastructures relying on different grid middleware (GRAM, CREAM, ARC and so on). It is out of the scope of this thesis explains their peculiarities, but how the selection of their resources can be constrained for a specific calculation. In this sense, it is customary to constraint the visibility of IM to providers that support certain VO or protocol. This is performed by setting the corresponding GLUE expression in the configuration of the IM, which will be used for querying the IS of the infrastructures.

However, the main mechanism to guide the selection or providers is introduced by the GWpilot PiF. In this chapter, only the basic features needed to straightforwardly run applications are commented. In this sense, users must take in mind that pilots created by the PiF are regular grid jobs, which are scheduled as usual by GridWay. Through the static configuration of the PiF, any user can establish preferences

```
...
<jsdl:Application>
       <jsdl:ApplicationName>TAIL_example</jsdl:ApplicationName>
       <jsdl-posix:POSIXApplication>
              <jsdl-posix:Executable>/usr/bin/tail</jsdl-posix:Executable>
              <jsdl-posix:Argument>-n</jsdl-posix:Argument>
              <jsdl-posix:Argument>100</jsdl-posix:Argument>
              <jsdl-posix:Argument>file.txt</jsdl-posix:Argument>
              <jsdl-posix:Output>stdout.${JOB_ID}</jsdl-posix:Output>
       </jsdl-posix:POSIXApplication>
</jsdl:Application>
<jsdl:Resources>
       <gw:GridWay xmlns: gw="http://www.gridway.org/gridway">
              <gw:RANK>CPU_MHZ</gw:RANK>
              <gw:LRMS_NAME>jobmanager-pilot</gw:LRMS_NAME>
              <gw:SUSPENSION_TIMEOUT>200</gw:SUSPENSION_TIMEOUT>
       </gw:GridWay>
</jsdl:Resources>
<jsdl:DataStaging>
       <jsdl:FileName>file.txt</jsdl:FileName>
       <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
       <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
       <jsdl:Source>
              <jsdl:URI>gsiftp://hostname/path/file1.txt</jsdl:URI>
       </jsdl:Source>
</jsdl:DataStaging>
<jsdl:DataStaging>
       <jsdl:FileName>stdout.${JOB_ID}</jsdl:FileName>
       <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
       <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
       <jsdl:Target>
              <jsdl:URI>gsiftp://hostname/path/stdout.${JOB_ID}</jsdl:URI>
       </jsdl:Target>
</jsdl:DataStaging>
...
```

(a) JSDL

```
(result, error) = drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, '/usr/bin/tail')
(result, error) = drmaa_set_vector_attribute(jt, DRMAA_V_ARGV, ["-n", "100", "file.txt"])
(result, error) = drmaa_set_attribute(jt, DRMAA_OUTPUT_PATH, ":stdout."+DRMAA_GW_JOB_ID)
(result, error) = drmaa_set_attribute(jt, DRMAA_GW_RANK, 'CPU_MHZ')
(result, error) = drmaa_set_attribute(jt, DRMAA_GW_REQUIREMENT, 'LRMS_NAME="jobmanager-pilot"')
(result, error) = drmaa_set_attribute(jt, DRMAA_GW_SUSPENSION_TIMEOUT, 200)
(result, error) = drmaa_set_vector_attribute(jt, DRMAA_V_GW_INPUT_FILES, ["file.txt"])
```

(b) DRMAA (Python)

```
NAME = TAIL_example
EXECUTABLE = /usr/bin/tail
ARGUMENTS = -n 100 file.txt
STDOUT_FILE = stdout.${JOB_ID}
REQUIREMENTS = LRMS_NAME="jobmanager-pilot"
RANK = CPU_MHZ
SUSPENSION_TIMEOUT = 200
INPUT_FILES = file.txt
```

(c) GridWay template

Figure 8.1: Description of a task with the different approaches offered by GWpilot.

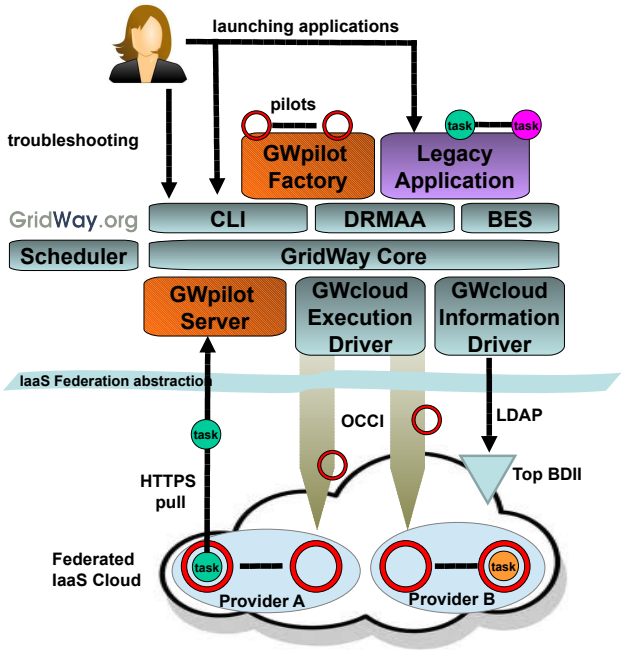


Figure 8.2: GridWay ecosystem architecture for cloud federations.

(*RANK*) and constraints (*REQUIREMENTS*) for all the pilots to dynamically select the proper resources in federations, but with the drawback of relying also on the IS.

Moreover, users can set a *SUSPENSION_TIMEOUT* for the pilot. In this case, its meaning concurs with the original one when the early binding approach is followed: the maximum waiting time allowed at remote queues. This aspect and the maximum number of pilots managed are the main parameters to be configured. However, PI, T and the maximum banned time are also important. Their influence on Task Scheduling was previously explained, and it is similar to Provisioning but considering that the discarded executions are pilots and the banned resources are the providers. Thus, T can be fitted to modify the discarding rate and to improve the suitability of resources selected, banning unreliable providers. On the other hand, PI always determines the task turnaround and it should not be modified due to Provisioning criteria unless GWpilot would be working on overloaded networks.

To improve understanding of differences between configuration parameters and similar statements used in tasks as well as their mutual influence, explanations are compiled in Table 8.1.

8.3.2. Provisioning in cloud federations

Provisioning in IaaS clouds can be easily achieved by means of the GWcloud drivers. The behaviour of the GWpilot PiS and PiF as well as the features introduced with GWcloud (see Chapter 5) are maintained when cloud resources are used. On the other hand, the implementation of pilots is lightweight and without library dependencies, i.e. they can run on any kind of Linux OS. So, no especial configurations are needed to deploy the pilot overlay on cloud federations, allowing users to choose their

Table 8.1: Differentiation between the meaning of basic task statements and configuration options as well as the effects on Workload Scheduling and Provisioning.

Task statements		**Provisioning configuration**	
-	It has influence in the T_{app} and T_{sched}	-n < *unsig. int* >	Maximum number of pilots
-	PI determines overhead in task dispatching and T_{app}	-i < PI >	Pilot pulling interval against PiS
-	$PI \cdot T$ determines when a task is killed by pilot because it does not communicate with PiS. Additionally, $PI \cdot T$ determines when the Scheduler considers a task as failed if no suspension timeout has been set for this task	-t < T >	Number of retries to communicate with PiS. $PI \cdot T$ determines when pilot ends because it does not communicate with PiS
$SUSPENSION\_TIMEOUT$	At least sightly higher than PI to avoid failing pilots. Sightly higher than $PI \cdot T$ to recover running tasks on unreliable network environments	-w < *unsig. int* >	Maximum waiting time in grid queues or for VM creation. (It is the $SUSPENSION\_TIMEOUT$ that PiF will set in pilot job declaration)
$REQUIREMENT$	At least $= LRMS\_NAME =$ "*jobmanager-pilot*" to run task on pilots only. Additionally other constraints can be set from Table 7.1, e.g. setting $QUEUE\_MAXTIME$ greater than the expected execution time of the task	-c <$REQUIREMENT$>	Expression to constraint pilot submission to certain providers. (It is the $REQUIREMENT$ that PiF will set in pilot job declaration)
$RANK$	Preference sets with numerical tags from Table 7.1	-r <$RANK$>	Expression to select more suitable providers for pilot submission. (It is the $RANK$ that PiF will set in pilot job declaration).
-	At least $PI \cdot T$ to force the end of an unfeasible running pilot	FR_MAX_BANNED	At least sightly higher than the -w option to discard unreliable providers

virtual environment. This distinguishing feature has not been achieved by the current pilot systems that follow a pulling mechanism [115, 31].

The approach followed is similar to the one used in grid, as can be seen in Figure 8.2. According to the number and requirements of the tasks created by any user or application, the PiF automatically submits the necessary pilots as usual, which are also stored in the Job Pool. Thus, the Scheduler is in charge of the matchmaking among cloud providers and pilots, the management of which is delegated to the GWcloud ED. This module works with pilots as common jobs. It wraps the execution of the pilots into the contextualisation files and performs the OCCI commands needed to boot the corresponding VMs. The started pilots will be continuously pulling PiS for tasks.

The basic configuration for Provisioning is the same than grid but with two exceptions. The existence of a contextualisation file, which is not needed to be modified for a straightforward adaptation, and the meaning of the $-w$ option, that is now related to the maximum allowed time to start the VM. Additionally, the banning feature is the only way to know the quotas set at providers, as has been previously commented in Chapter 5.

Therefore, users can run their legacy codes on GridWay as usual. The tasks created by these applications are scheduled among the pilots enrolled and within the virtual environment selected. To better understand how the Task Scheduling and Provisioning are coordinated in a cloud federation, the sequence of steps that compiles the ones described in Figures 5.2 and 7.5 is explained as follows and in Figure 8.3:

1. GWcloud ID works as in Subsection 5.3.3. It periodically searches for cloud provider updates at top-BDIIs, which are included into the Host Pool.

2. The GridWay Scheduler notices that some cloud provider is *free* and fulfils the requirements of certain pilot waiting in the Job Pool. Consequently, the *SUBMIT* operation is sent to the GWcloud ED.

3. The GWcloud ED processes the operation as in Subsection 5.3.3, sending back a *CALLBACK* operation and performing the *create* operation against the OCCI service of the provider.

4. The provider creates the VM following the provided *os_tpl* and *resource_tpl*. Through the booting process, the Cloud-Init contextualisation starts the pilot job.

5. The pilot advertises the PiS and updates their characteristics. Therefore the Provisioning phase is completed and begins the Task Scheduling as usual (see Subsection 7.2.2). GridWay Scheduler continuously dispatches tasks to the pilot if they are waiting in the Job Pool.

6. When the pilot is *idle* during the $T \cdot PI$ interval, it *ends*, and as any other job, the VM automatically shuts down.

7. The GWcloud ED periodically tests the VM state through OCCI operations. If the VM shuts down, the driver performs the deletion of the virtual workspace because the pilot has *ended* its execution.

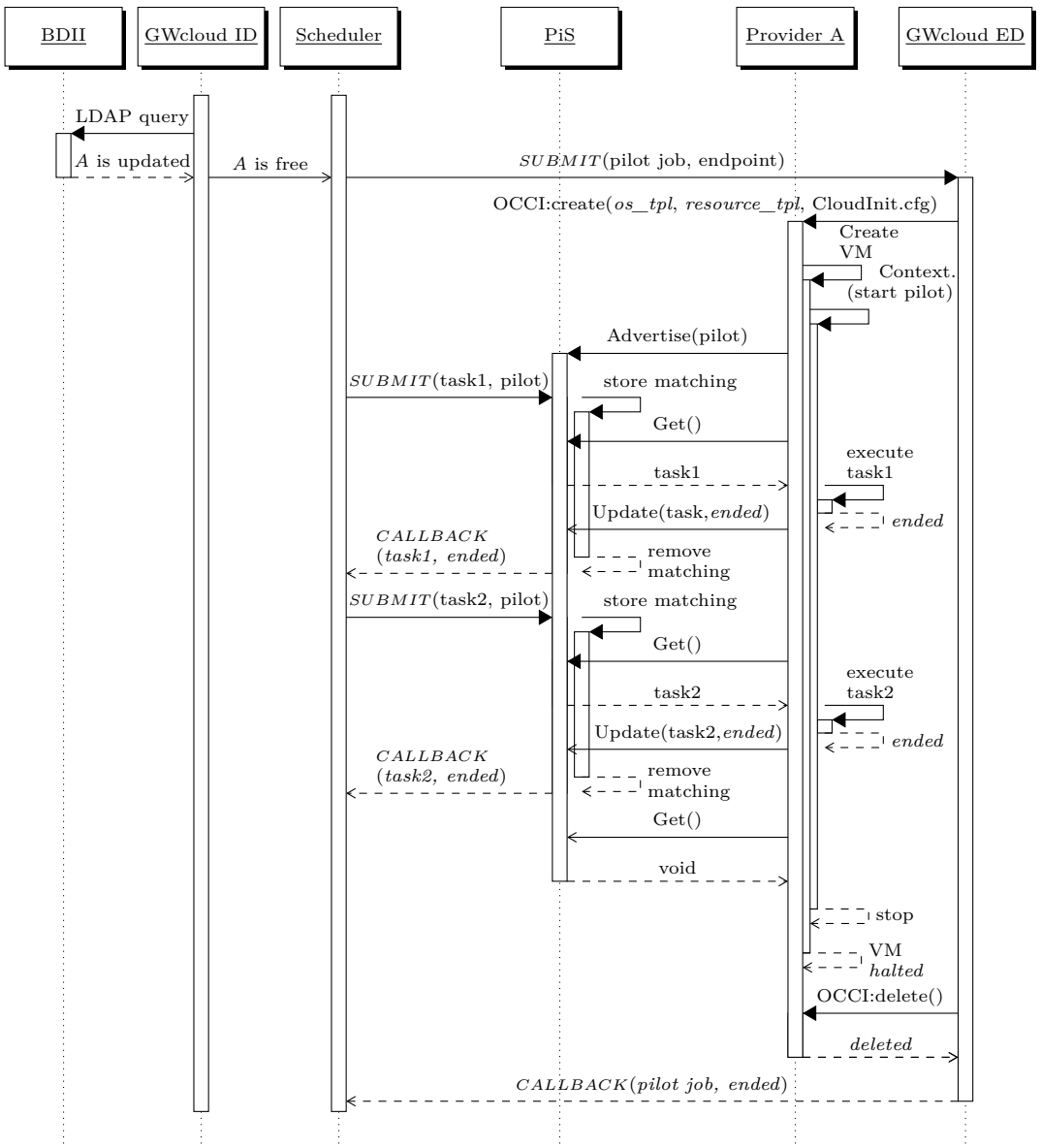


Figure 8.3: Sequence of activities performed by the actors of GWpilot to accomplish any task in clouds. They correspond to the steps 1-7 described in Subsection 8.3.2.

8.4. On-demand radiotherapy simulations on the cloud

The objective of this Section is to show a simple, but real use case that focused on customising cloud provision, also profits from federated clouds. The proposed experiments will demonstrate the basic capabilities of the solution that differentiates from other approaches, such as decentralisation, middleware independence, capacity for accomplishing short tasks, on-demand Provisioning, compatibility with legacy applications, dynamic brokering, etc.

For this purpose, a real application that should be managed in a personal workstation is distributed and executed on-demand in the FedCloud resources. Additionally, the idea is not to set up a pre-configured VM or a VM similar to the worker nodes used in the EGI grid infrastructure, i.e. Scientific Linux with gLite/UMD. Therefore, a clean Debian-based template image will be chosen from *appdb.egi.eu* repositories.

8.4.1. Legacy application and configuration

BEAMnrc [71] is a general-purpose Monte Carlo code to simulate the radiation beams from radiotherapy units. Since the outcome of Monte Carlo simulations is based on random sampling, typically $\sim 10^8$ particle histories are needed for good accuracy, taking weeks of computation on a ~ 2 GHz processor. This remains the main hindrance in clinical implementation of Monte Carlo simulations. For this experiment it has been used a typical calculation of $4 \cdot 10^8$ particles and a rectangular geometry, which represents a use case that can be approached in a hospital environment. The workload was divided into 2,000 tasks, which lasts between 150-280 s on current processors (2.4-3.2 GHz).

The idea is that the specialist in radiotherapy will launch the simulation on his own workstation before starting any treatment. Therefore, a machine with one i3-530 (2 cores, 2.93 GHz) and 4 GB of RAM was configured with GridWay, GWpilot and GWcloud drivers. Additionally, to avoid the necessity of host certificates, the system will use GASS as transference protocol. The *fedcloud.egi.eu* VO belonging to the EGI FedCloud infrastructure is used to perform the tests.

To ease the comparison with the previous experiments, and because one of the objectives is to create Debian-based VMs in FedCloud, the configuration of GWcloud is maintained, being Table 5.3 valid again to describe the suitable providers. For the same reason, the configuration options set for GWpilot in Section 7.4 are maintained, except those related to cloud Provisioning. In this sense, the PiF is allowed for managing a maximum of 200 pilots (running on VMs), but the Scheduler only will wait 600 s for the creation of every VM. On the other hand, the dispatching chunk, i.e. the number of tasks and VMs managed during a scheduling cycle of 10 s, is set to 20. However, the submission is also limited to dispatch one VM per suitable provider in every cycle.

Obviously, the resource banning feature of GridWay is enabled, so whenever a pilot fails, its hosting provider is banned for a variable period of time. This last feature will be of importance in the experiments, because the only way to currently know the quotas established at providers is by continuously testing the creation of VMs.

Note that BEAMnrc was already adapted using the DRMAA producer-consumer library described in Section 4.3. Thus, to enable application to profit from Task Scheduling supported by GWpilot, only setting the requirement argument to $LRMS\_NAME$ = "*jobmanager-pilot*" is needed.

Therefore, any unskilled user can easily perform all those configurations.

8.4.2. Results

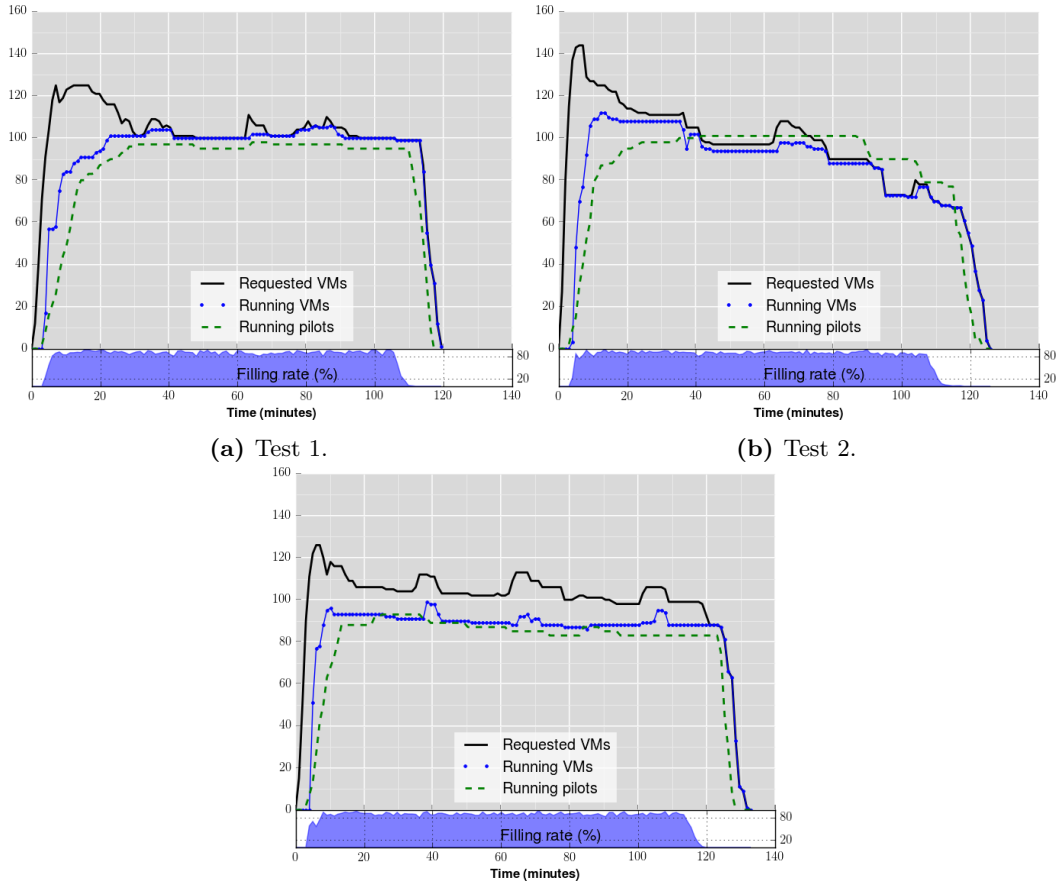
Three identical tests as previously described have been performed in different days to avoid the influence of unusual infrastructure statuses.

First outcome is the noteworthy limitation of the amount of available resources. The number of reliable providers is really smaller (7) than the number of theoretically suitable providers (14), as it is shown in Table 8.2. These results demonstrate the two affirmations done in this work: as grid sites, cloud providers are not immune from errors produced by the middleware, network outages or misconfiguration; the other issue is the illusion of the fully availability of the resources, because user quotas are not currently shown in information systems. Therefore, experiments demonstrate the suitability of proposed Provisioning based on continuously checking the real availability and reliability of every resource. Scheduler always dispatches a VM to providers until it gets a failure from the GWcloud Execution driver. Consequently it bans the provider during $3{,}600 \cdot (1 - e^{\Delta t/650})$ s, i.e. according to the elapsed time Δt from last failure, the banning time can be set to a maximum of one hour. These are the reasons for the number of failed creations in every reliable resource as they correspond to the times that the driver has failed to create a VM because any quota has been reached. In the case of unreliable resources, this number really means the number of failed VM creations and OCCI errors.

Same aspects can be seen in Figures 8.4-(a, b, c). They show the evolution of Provisioning through the three tests from the user's point of view. The number of VM requests is higher at the beginning because all suitable resources are tested. Then, some of them are actually started. Scheduler progressively takes account of failed attempts, and the gap between requested and running VMs is progressively reduced. However, not all started VMs correctly perform the contextualisation or provide network connectivity (at least through NAT). Additionally, VMs last in creation (bear in mind that 600 s are allowed) and booting (starting services and contextualisation), so pilots try to connect to GWpilot server (20 tries $\cdot$ 30 s = 600 s) until they end. Moreover, VMs can unexpectedly crash. These are the reasons of the difference between pilots registered and the VMs running, in special for the Figure 8.4-(b), where some VMs fail on the 76^{th}, 90^{th} and 108^{th} minute, but the system last in discarding the pilots the corresponding 600 s.

The slope in the appropriated resources is pronounced as expected [65]. This capability differentiates among other mechanisms of provision in pilot systems that are not able to manage together several Provisioning requests. In this sense, nearly the maximum of provisioned VMs is reached in around 20 m and the half before 10 m. These measurements are mainly related to the VM creation time limit of 600 s. Therefore, it is reasonable to obtain the most provisioned resources after the first failed creation interval. From this 20^{th} minute point on, the number of VMs remains roughly stable. Thus, it can be concluded that the maximum usage limits of the cloud infrastructure is reached by the user launching the application.

It is important to mention how the system makes the most of the provisioned resources. For this reason, the filling rate of pilots with execution tasks is added to Figures 8.4-(a, b, c). The average is above 85 %, which is a great result that guarantees an efficient profiting when short tasks are scheduled.



(a) Test 1.

(b) Test 2.

(c) Test 3.

Figure 8.4: Resource Provisioning during the execution of the BEAMnrc application on the EGI FedCloud infrastructure. Additionally, the filling rate is included for every test.

Table 8.2: Number of VM instances successfully set up and failed at every provider. Additionally, the number of failed requests at unreliable sites is shown.

	Test 1		Test 2		Test 3	
Provider	set up	failed	set up	failed	set up	failed
https://carach5.ics.muni.cz:11443	18	6	11	5	10	4
https://controller.ceta-ciemat.es:8787	5	7	15	4	27	3
https://egi-cloud.pd.infn.it:8787	30	13	24	5	24	4
https://fc-one.i3m.upv.es:11443	4	4	-	-	13	8
https://nova2.ui.savba.sk:8787	30	3	19	3	19	6
https://prisma-cloud.ba.infn.it:8787	2	3	22	10	0	8
https://stack-server-01.ct.infn.it:8787	12	3	11	6	10	4
Unreliable (7)		104		92		109

8.5. Conclusions

The achievements introduced in this chapter consolidate GWpilot as a general-purpose pulling pilot system that successfully works on grid and cloud environments. Among the demonstrated features compiled there are a set that differentiates the framework from other approaches:

- Friendly interface and compatibility with legacy applications.
- Efficient execution of very short tasks.
- Independent and easy configuration.
- Lower overheads that allow decentralised and local installations, even on the PC of the user.
- Discovery and selection of providers according to preferences and constrains, even on cloud federations.
- Personalised virtual environments.
- Post-configuration of VMs on demand.

Therefore, through GWpilot and their drivers, any legacy application can improve its performance by the straightforward use of the late-binding design. Unlike other approaches, this framework can be easily deployed and customised and it is able to perform a dynamic Provisioning based on the specific status of grid and cloud federations at any time.

Although the basic support of Task Scheduling and Provisioning has been shown, the performance when these layers are combined has not been evaluated yet. Additionally, the configuration options described makes Provisioning too static and tied to IS and to individual calculations.

Furthermore, as the use of GWpilot includes all the advantages summarised in the beginning of Section 7.2, the system can potentially support the customisation of whole scheduling at user-level, allowing among other features:

- Personalised monitoring of new configurations, performed either in provisioned grid WNs or in VMs from the cloud.
- Advanced Task Scheduling and Provisioning techniques, preserving the fair-share and making use of several infrastructures.
- Capacity of support other scheduling tools such as self-schedulers.

These achievements jointly with their experimental demonstration are performed through the following chapters.

Chapter 9

Improved Scheduling and Provisioning Techniques

9.1. Introduction

The GWpilot design provides users with high levels of performance as was demonstrated through the previous Chapters. First, the GWpilot Server uses its MAD role to notify and receive GridWay messages about pilot and task statuses in a continuous stream. Second, the size of messages was reduced to a minimum in all the communication stages. Thus, the message protocol used is a simple HTTP request from the pilot, which is less complex than the one used in web services, for example. Additionally the information is formatted in the tag-based mechanism of GridWay, which is simpler than other specification formats, such as the ClassAds [259] language used in Condor. With these decisions, the objective is two-fold: the reduction of the task turnaround time through the limitation of overheads, and the possibility of adapting the behaviour of the system by introducing customised policies to perform a specialised scheduling for applications with certain computational needs.

Furthermore, the GWpilot system is multi-application and multi-user, maintaining the fair-share policies of GridWay. These features allow better profiting from remote resources appropriated by pilots because different task types can be scheduled to reduce the number of idle nodes. Additionally, statistics from pilots, resources and users are dynamically compiled and can be subsequently used in other executions. These are important advantages over application-oriented pilot frameworks, such as DIANE or BigJob. Moreover, GWpilot is able to concurrently provision resources belonging to different DCIs.

However, when the users, tasks, providers, pilots and characteristics taking into account for scheduling exceed certain volume, the performance achieved by the GridWay Scheduler falls into unfeasible values. It is noteworthy to mention only one instance of the Scheduler performs whole matchmaking. This is so due to GridWay has initially developed for early-binding scheduling in a grid environment, where the number of resources is fewer than the ones managed with pilots. Moreover, cloud providers increase the complexity by adding different characterisation and behaviours. Additionally, the capacity of overloading grid queues was not implemented because this feature is usually counterproductive in early-binding. On the other hand, mana-

ging whole scheduling at the same box as was required in Subsection 3.2.2, will enable the system to perform Multilevel Scheduling. This achievement will be properly justified in Chapter 10, but they are consequence of the work performed through this one.

To solve these issues, modifications in the algorithm of Scheduler, but also in the GridWay core and the Information Manager (IM) have been performed to improve its performance to the levels required. Additionally, it was implemented a smarter method for queuing pilots at remote grid sites, meanwhile the generality needed for an efficient cloud Provisioning is preserved. Therefore the objectives of this chapter are to:

1. describe the matchmaking algorithm of the Scheduler and the improvements made;

2. demonstrate that multiple applications can run together on the system and can implement more advanced Workload Scheduling than FCFS.

3. show the multi-DCI capacity with the use of multiple execution drivers at same time;

4. prove overload introduced does not induce added overheads that modify turnaround.

9.2. Improved matchmaking

A common execution with GWpilot demands a performance level from GridWay that has never been tested before. First, the number of pilots that can be provisioned from current DCIs clearly exceeds several thousands. As an example, the experiments performed in Chapter 7 demonstrated the capacity of GWpilot for Provisioning more than one thousand pilots from a medium-sized VO belonging to the EGI federation. The GridWay core was originally designed to support high loads and can easily manage multiple MADs and these volume of jobs, but its Scheduler module does not response in the time order required to accomplish short-tasks.

The evaluation of this new volume of pilots and the corresponding tasks create a computational complexity that is infeasible for the Scheduler if its original algorithm [10] is used. The algorithm is composed by two main loops, where the first one evaluates the suitability of every resource for every job, and the last one performs the job dispatching following the classification previously performed, but observing the fair-share rules configured.

In general, the first main loop compares the constraints of every job (j) with the characteristics of every queue (q) to select an adequate one. Subsequently, it performs the ranking of these filtered queues following the description of the job. Thus, the complexity of this procedure is close to $\Theta(Q \cdot J)$, where Q is the set of queues offered by resources (i.e., hosts: H), and J is the number of jobs waiting to be scheduled. The number of parameters evaluated also influences this estimation, but its order of magnitude is incomparable with Q and J and can be discarded for the estimation.

In the case of GWpilot, J is the addition of the tasks and the pilots waiting in the system because both types are considered at the same time by the Scheduler. On the other hand, Q also comprises the virtual queues offered by the pilots already

provisioned. Therefore, if GWpilot maintains $2 \cdot N = Q + J = 10,000$ items, the complexity is close to $\Theta(N^2)$, and the scheduling will last on the order of minutes.

To guarantee the performance of the system, the part of the Scheduler that performs the initial matching of tasks-pilots-resources is modified to achieve a complexity of $\Theta(\log(Q) \cdot log(J))$, as it is shown in Algorithm 2. The suitability of this solution will be demonstrated in Section 9.5.

Algorithm 2: More efficient selection of resources, avoiding to evaluate the similar jobs twice (i.e. tasks and pilots for GWpilot case), and based on the new estimation of slot availability at remote queues.

$Q \longleftarrow \{h.q\ (\forall h \in H)\ \};\ \ S \longleftarrow \{\emptyset\};$
for $j \in J$ **do**
  $j.Q \longleftarrow \{\emptyset\};$
  if $\nexists(q''.requirements \in S = j.requirements)$ **then**
    for $q' \in Q$ **do**
      if $Fulfil(j.requirements, q'.characteristics)$ **then**
        if $q' \notin S$ **then**
          $q \longleftarrow (copy)q'$
          $max\_job\_running \longleftarrow$ mín$(q.$MAXRUNNINGJOBS$, q.$NODECOUNT$)$
          $max\_jobs \longleftarrow$
          máx(mín$(q.$MAXJOBS$, q.$NODECOUNT $\cdot\ 2)), max\_job\_running)$
          $q.max\_wait \longleftarrow max\_jobs - max\_job\_running$
          $q.queueable \longleftarrow$
          máx$(0, max_jobs - q.$JOBRUN $- q.$JOBWAIT $- h.used\_slots)$
          $gwfreenc \longleftarrow$
          mín$((max_job_running - h.used_slots), q.queueable)$
          $q.slots \longleftarrow$ mín$(gwfreenc, q.$FREENODECOUNT$)$
          $q.requirements \longleftarrow j.requirements$
          $q.rank \longleftarrow ComputeRank(q.characteristics, j.rank);$
        else
          if $\exists(q''.rank \in S = j.rank)$ **then**
            $q \longleftarrow (copy)q''$
          else
            $q \longleftarrow (copy\ any)q'' \in S$
            $q.rank \longleftarrow ComputeRank(q.characteristics, j.rank);$
          $S \longleftarrow S \cup \{q\}$
          $j.Q \longleftarrow j.Q \cup \{q\}$
  else
    for $q' \in S$ **do**
      $q \longleftarrow (copy)q'$
      if $j.rank \neq q.rank$ **then**
        $q.rank \longleftarrow ComputeRank(q.characteristics, j.rank);$
        $S \longleftarrow S \cup \{q\}$
      $j.Q \longleftarrow j.Q \cup \{q\}$

9.3. Overloading queues

Alternately, pilot systems should benefit from the slot appropriation to obtain better resources, although pilots have to wait long times at remote queues. This aspect was not fully explored in previous GridWay releases because it was not needed for the scheduling based on an early-binding approach. Therefore, a new improvement is necessary to properly fill remote queues with waiting pilot jobs, but without overloading them with unproductive pilots that will never be executed.

Nevertheless, IS do not completely publish all the information related to the usage of remote LRMS queues. Following the GLUE [5] scheme, IS only show the total capacity ($NODECOUNT$), free slots ($FREENODECOUNT$), running jobs ($JOBRUN$), waiting jobs ($JOBWAIT$), the maximum available slots per VO ($MAXRUNNINGJOBS$), and the number of jobs that will fill the queue ($MAXJOBS$) if they are completely free. Note that this information could be even erroneous. No information about user priority or the number of remaining jobs per VO is given. Obviously, the number of jobs that a specific user should submit to a queue is neither the maximum queue capacity nor the published free slots because the LRMS fair-share policies usually imply that only a few of these nodes are really available for a non-privileged VO user. All of them constitute a problem for any Provisioning algorithm because it is impossible to assure the slot availability in any site. The use of pilot jobs can reduce the influence of this problem because a bunch of pilots can be launched to every remote queue. However, the number of pilots must be limited to avoid overloads and the possibility of being banned from sites. To attenuate this issue, pilot systems usually take into account only the free slots or implement their own Provisioning algorithms but do not provide users with mechanisms to customise this behaviour.

For this purpose, the GLUE tags are monitored and included as generic variables in GridWay to be used in scheduling decisions. Because these GLUE tags could contain inaccurate information, two separate estimations are made: the maximum number of jobs allowed in the waiting state (max_wait), and the number of jobs that are allowed to queue ($queuable$) at that moment. They are included in Algorithm 2, which evaluates the resources with respect to the description of tasks.

Nevertheless, these estimations will not be absolutely correct because any estimated value will most likely be higher than the correct one, even more so in a highly dynamic environment such as a grid. For this reason, a new configuration parameter was introduced into the GridWay configuration that indicates the percentage (σ) of the maximum number of waiting jobs (max_wait). Additionally, foreseeing an over-estimation, the scheduler will now use $usable\_slots$ as a global prediction at the moment of dispatching pilots to resources. This procedure is depicted in Algorithm 3.

9.4. Scheduling configuration and performance

Items belonging (J) are ordered according to the initial priority given by the user, the weight of which progressively increases as j remains waiting. However, although it is not expressed in Algorithms 2 and 3, every user (u) and resource (h) has assigned some priority that is used to build the final priority of either the task or the pilot (j). Thus, algorithms summarise how the fair-share is performed in GWpilot and how the limitations set ($DISPATCH\_CHUNK$, $MAX\_RUNNING\_USER$ and $MAX\_RUNNING\_RESOURCE$) has influence on task and pilot dispatching. For indi-

Algorithm 3: New dispatching algorithm improved for filling queues. It is performed every *SCHEDULING_INTERVAL*.

```
dispatched ⟵ 0
for j ∈ J (ordered by j.priority) and dispatched < DISPATCH_CHUNK do
    u ⟵ j ↑ user
    running_user ⟵ u.used_slots + u.dispatched
    if (running_user ≤ MAX_RUNNING_USER) then continue;
    for q ∈ j.Q (ordered by q.rank) do
        h ⟵ q ↑ hostname
        running_host ⟵ h.used_slots + h.dispatched
        if (running_host ≥ MAX_RUNNING_RESOURCE) then continue;
        free_slots ⟵ q.slots − h.dispatched
        queueable_slots ⟵ q.queueable − h.dispatched
        usable_slots ⟵ mín(q.max_wait · σ · 0,01 + free_slots, queueable_slots)
        if usable_slots ≥ j.np then
            dispatched ⟵ dispatched + 1
            h.dispatched ⟵ h.dispatched + 1
            u.dispatched ⟵ u.dispatched + 1
            gw_scheduler_job_dispatch(j, q)
            break
h.used_slots ⟵ h.used_slots + h.dispatched
u.used_slots ⟵ u.used_slots + u.dispatched
```

vidual users, the interest is in the possibility of prioritising some task over others. This feature is used in the experiments described in Subsection 9.5.3 to enable scheduling polices among applications.

9.5. A more reliable mechanism to perform transport calculations

The intention is to show how the GWpilot configuration and the parameters of a legacy application can be adjusted to improve the final performance and reliability of a real calculation, in this case, the DKEsG and ISDEP codes adapted in Chapter 4. For this purpose, the validity of the assumed ordering of DKEsG in low collisional plasmas was explored. The importance for the fusion community, the underlying physics and the interpretation of the physical results obtained from the executions performed in this section are fully explained in the Appendix C.

Furthermore, for the computational field, the objectives of the calculations are multiple. First, the aim is to demonstrate that several legacy applications can straightforwardly run on GWpilot at same time and profit from its advantages, in particular, implementing Workload Scheduling different to FCFS without experiencing performance losses due to the Scheduler. Another main interest is to introduce the significance of the measurements performed in this thesis and to subsequently show the reduction of queue waiting times and middleware overheads when GWpilot is used. For the sake of comparison, the results obtained in Chapter 4 will be taken into consideration, especially those regarding the turnaround overhead and job failures. With relation to this, another objective is to demonstrate that important overheads can be estimated based on the selected configuration, especially the ones generated by

Scheduler. Additionally, it is important to show how GWpilot can establish a network overlay among different type of resources, for example, the accesses by CREAM and the GRAM middleware.

9.5.1. Test bed

The calculations have been performed on resources belonging to the GISELA/S-CALAC infrastructure. For this purpose, GWpilot on a GridWay 5.8 with the improvements described through previous sections was configured on a Scientific Linux 5.7 that contains the default Python modules for the distribution and the gLite 3.2 client middleware. Thus, GWpilot schedules all the generated pilot jobs on the *gisela* VO (i.e. *prod.vo.eu-eela.eu*). It is noteworthy to mention that GISELA is the extension of the EELA-2 project and its grid infrastructure is similar to the one studied in the Section 4.4. To make the most of the newly implemented scheduling features and the pilot jobs, in this experiment the maximum resource allocation is restricted to $MAX\_RUNNING\_RESOURCE$=100 slots per site, discarding CEs that offer 32 bit WNs. Additionally, the backup CEs and CERN resources are also discarded. The resulting available computational power measured in slots for this VO is shown in two last rows of third column in Table 9.1. GridWay is also configured to allow submitting $\sigma = 15\,\%$ more jobs than the estimated free slots (but not over the established limit of 100 jobs). While GWpilot is configured to create a maximum of 550 active pilots, the dispatching suspension timeout (the maximum time spent at the remote LRMS) for these pilot jobs is set to 5 hours, and the pulling interval is established in 45 seconds with a maximum number of retries of 20.

Despite ISDEP and DKEsG already has been adapted to use CLI and both run on GridWay, the work presented in Chapter 4 only demonstrates that users can perform executions in a grid following the early-binding mechanisms, but not with pilots. To properly benefit from GWpilot, these programs have been straightforwardly adapted according to the Subsection 8.2 to provide another parameter option that automatically includes the $LRMS\_NAME$ = "*jobmanager-pilot*" requirement when BoTs are submitted. This allows users to choose using the pilot system. Additionally, they allow declaring any ranking or requirement expression as a parameter to benefit from the reliable characterisation of pilots. It is important to explain again that the suspension timeout acquires a new significance when pilots are used, i.e., it is the response time from the pilot when a BoT is scheduled to it. Therefore, the user has control of the important configuration aspects that influence the performance of the computation, as will be shown in the test described through this Section.

Moreover, as the intention is to improve the compatibility with legacy applications, DKEsG and ISDEP have been adapted to DRMAA following the producer-consumer library described in Section 4.3. In this sense, the following configuration has been set for DKEsG and ISDEP: the maximum number of submitted BoTs has been limited to 500; the $SUSPENSION\_TIMEOUT$ (the maximum time for fetching the task by the pilot, as commented in Subsection 8.2) has been set to 60 seconds; the resource allocation priority has also been set relying on the CPU speed, as GWpilot does (not regarding the corresponding IS, but the real characteristics notified by pilots using the $RANK = PILOT\_CPU\_MHZ$ expression); and the polling time has been set to a quarter of a second.

Therefore, it is worth mentioning that the number of DKEsG instances in the GridWay queue will not reach the maximum number of pilots that could be available

in a certain time. This is done to improve the resource usage as time goes by: the best pilots will be selected, and the rest will die whenever no work is assigned to them. Additionally, this over-submission of pilot instances is set to allow a certain error margin in the estimation of usable slots, especially when queues are overloaded. This circumstance is shown in Table 9.1, where the maximum number of usable slots for these 550 pilots is only 500.

9.5.2. Preliminary test

DKEsG-Mono executions have highly variable completion times depending on one parameter under consideration, the plasma radii. Additionally, other input parameters, such as the selected Legendre and Fourier polynomials, also influence the duration of the task, which is affected by an exponential order. Therefore, it is mandatory to run a lightweight test to subsequently obtain adequate DKEsG-Mono parameters in the ranges that successfully ensure the accurateness of the final results because the main computation can last for weeks. This preliminary work is not necessary for ISDEP because the particle trajectories are independently calculated in a much lower time, as will be explained below. Subsequently, the main calculation is performed to compare the accuracy of the ISDEP and DKEsG results and it is described in Subsection 9.5.3.

For accomplishing a superficial calculation with DKEsG, a typical parameter scan range corresponding to the TJ-II fusion device [260] was selected for DKEsG-Mono. This implies a broad variation of three input parameters: the collisionality (ν); the radial electric field (E); and the radial position of the plasma (ρ). The two first parameters are included in $CMUL$ and $EFIELD$ parameters, as was explained in Section 4.4. The ranges of these parameters are finally set for this work as follows:

$$\rho[0.00714,\ 0.0143 \cdots,\ 0.993,\ 1],\ \text{EFIELD}[\text{-}250 \cdots\ 250{:}10],$$
$$\text{CMUL}[log(1 \cdots 9)10^{(-7 \cdots 0)}]$$

resulting in 140 radial positions, 51 values of the radial electric field and 72 values of the collisionality.

This input parameter combination results in 514,080 independent calculations, whose CPU consumption is directly proportional to the plasma radii considered. Note that the most updated DKES [245, 248] code is used now. In this sense, that consumption ranges from 58 to 751 seconds on an old 64-bit processor Intel Xeon X5365 3 GHz and from 40 to 523 s on a recent Xeon E5-2667 2.9 GHz. The first machine is taken as a lower reference of the performance required by the application for enabling comparisons with previous experiments performed in Section 4.5. Additionally, to perform a similar test with respect to the one in Subsection 4.5.2, five calculations were grouped in a unique BoT.

Subsequently, the resulting output data are used by the DKEsG-Neo module to calculate the Neoclassical transport coefficients. The objective now is to obtain NC coefficients that allow the calculation of fluxes. Thus, the average time spent for any of these 420 new tasks is approximately 20 minutes and 12 minutes on the aforementioned machines, respectively. Therefore, they could be directly added to the whole computation without any chunking. Thus, the experiment is finally composed of 103,236 independent BoTs, the submission of which will be ordered from inner to outer radial positions in the plasma, i.e., the farther outward the radial position, the longer the execution time.

Table 9.1: Executions of DKEsG bags of tasks in pilots and the corresponding grid jobs containing pilots on GISELA infrastructure.

Resources			Executions					
Type	Max. num.	Max. usable slots	Submitted	Total	OK	Failed	Err.	Susp.
Pilots	550	500	BoTs	103,628	103,236	392	177	215
CREAM	$15^{(a)}$	$672^{(b)}$	Pilots	3,479	1,146	2,333	882	1451
GRAM	$5^{(a)}$	$402^{(b)}$	Pilots	1,957	938	1,019	621	398

$^{(a)}$ Discarded 32 bits WNs, backup CEs from the same site and CERN resources.
$^{(b)}$ Limited to 100 slots per resource.

Results

The time spent by the preliminary test was 94 h 27 m 31 s. For the sake of comparison, if this calculation were performed sequentially on the selected machines described in the previous section, it would have lasted for approximately 6.58 and 4.54 years, respectively, i.e., 610 and 420 times slower than this GWpilot-enabled test, where usage is limited to 500 pilots.

Table 9.1 shows the comparison between the resultant executions of BoTs using pilot jobs and the corresponding grid jobs utilised for wrapping these pilots. The column labels stand for the ones also set in Table 4.3 to allow the comparison: BoTs (or pilot jobs) successfully executed, which are grouped under the *OK* column, and for the total number of failed BoTs (or pilot jobs), which appear under the *Failed* column. The last two columns depict the failures due to an exceeded suspension time at remotes resources (*Susp.*) and the ones produced by any other error (*Err.*). Note that neither performance thresholds that would stop pilots or tasks nor migration is allowed, unlike the experiments in Section 4.5. However, the suspension timeout for pilots and tasks have been set.

First, a distribution of jobs (the pilots) among CREAM and GRAM resources can be appreciated with a high incidence of failed jobs with respect to the grouped tasks executed by pilots. The higher number of suspension errors in CREAM sites are related to their overload: there are many resources that offer few slots (< 100). This implies more queue waiting time, so many jobs were discarded before the 5-hour threshold. It should be noticed that although the accumulated time wasted by suspended pilots was 1 year and 20 days, the total time wasted at remote queues by successfully executed pilots was 14 days. Additionally, the number of waiting pilots has been maintained roughly constant at a value slightly less than 50, with the exception of the moments in which GridWay overhead is increased. These exceptional circumstances will be explained below, but the remaining results indicate that the new mechanisms described in Section 9.3 are suitable for Provisioning pilots. Thus, GWpilot masks the negative influence of these overheads, which are not appreciable by the user.

The number of failed BoTs in the pilots' execution deserves a fuller explanation. As the reader can appreciate, although GWpilot is much more stable than the traditional early-binding methods, it is not immune from errors produced by the middleware, network outages or any other typical problem related to the grid. This is the case of the 392 failed executions shown in Table 9.1, which are the consequence of pilots dying when a BoT is running inside (*Err.*) or are produced whenever a group of tasks has been dispatched to a dead pilot that has not been discarded yet. Nevertheless, both sources of error only represent four per mil of the total submitted executions.

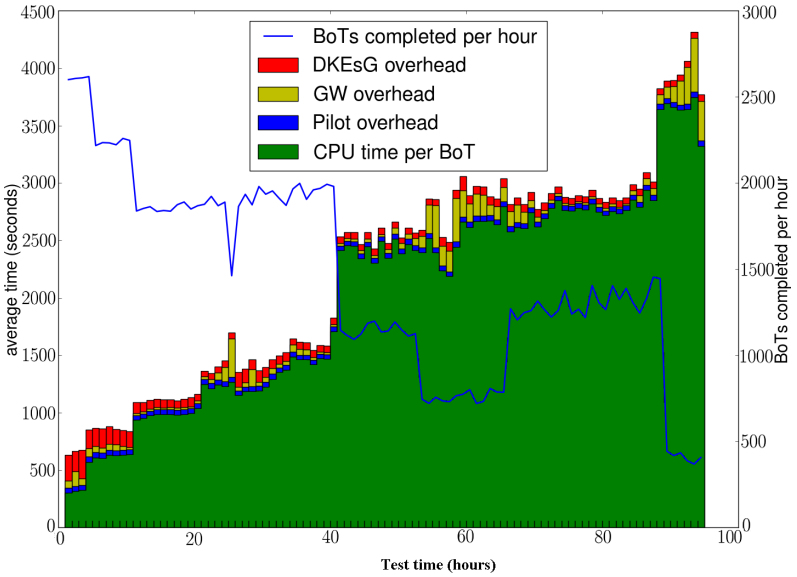


Figure 9.1: Turnaround average times per hour of BoTs submitted by DKEsG to GWpilot system. It is compound of the real consumed CPU time at remote resources, the GridWay scheduling time and the added pilot overhead. The time for generating BoTs and their completion rate are also displayed.

However, one objective of the test carried out is to demonstrate better performance than that offered by traditional submission mechanisms for the grid. For this reason, the real turnaround time of each BoT is measured, i.e., the time difference between when a BoT is queued for dispatch in GridWay and the GridWay acknowledgement that the BoT is completed. This value can be decomposed in the real consumed CPU time at the remote resource, the overhead produced by the pulling interval performed by pilots, and the overhead primarily produced by the incapacity of the GridWay scheduler to dispatch BoTs due to the inexistence of free resources (in this case, idle pilots). These three average values per hour are represented in Fig. 9.1.

Although it is not concerned with GWpilot, the DKEsG overhead is also displayed in the figure because it influences the final makespan of the calculation if not enough DKEsG instances are provided to fill all the available pilots. This circumstance is shown during the first hours of the test, when more successful BoTs than those that DKEsG can manage are returned because they have very short duration (250 to 650 seconds). Later, for the remaining calculation, DKEsG overhead remains stable. Pilot overhead is always between 41 and 43 seconds because it only depends on its configured pulling interval, thus demonstrating the scalability of the GWpilot system, which is not influenced by the number of pilots running.

Because the maximum number of usable pilots is limited to the same maximum volume of BoTs, an increase of the GridWay overhead indicates that the number of running pilots is less than 500. This fact explains the high values compiled for the test intervals from 23 to 28 hours, from 52 to 68 hours and from 89 hours to the end of the calculation, where many pilot jobs were ended because of the remote queue limitation time (mostly 24 or 48 hours of consumed CPU time). In this final interval, more pilots were also submitted, although most of them were queued, waiting to be

executed.

The blue line in Fig. 9.1 shows the number of completed BoTs per hour, and its value decreases as time passes because the time consumed by each BoT increases as the test progresses due to the submission of BoTs being ordered from inner to outer radius. Dales also indicate that the number of available pilots decreases. These two circumstances are especially relevant beyond the 89 hours of the calculation: in this case, the duration of a BoT is longer than one hour, and the system can only run a number of pilots below 500.

As an additional conclusion to the previous commented makespan value, it can be mentioned that the accumulated turnaround overhead only represents 6.21 % of the return time of every completed BoT. This value supposes an impressive performance improvement from 61.48 % compiled in Subsection 4.5.2 because it does not also comprise failed jobs.

9.5.3. Main computation

Maintaining the same configuration for the GWpilot framework, the interest now is to accomplish a computation that allows the comparison between the resultant DKES and ISDEP fluxes but reduces overheads to a minimum. To ensure accurate results in DKEsG, the number of nonzero Fourier coefficients has been increased, and as consequence, the consumption times of the DKEsG-Mono task ranges now from 20 minutes to 27 hours. Meanwhile, every ISDEP task calculates 10 trajectories and lasts approximately 100 s each. Both time measurements have been done on the reference Xeon X5365 machine because it is again necessary to know how suitable the execution of these calculations on the lowest hardware is. However, a modern processor such as the powerful reference would halve the execution time. Thus, the task grouping is not necessary for the DKEsG-Mono tasks because the minimal duration of a task is longer than 750 s, and therefore, the application overhead will be fixed to the minimum. Additionally, the maximum wall-time of one task is expected not to exceed 48 hours. However, ISDEP tasks must be grouped into BoTs, i.e., a chunk of 20 tasks is performed for this calculation.

To compare the results provided by DKEsG and ISDEP, it is only necessary to select a set of ρ. This will noticeably reduce the entire makespan but not the variability of the execution times of every task. Then, the new DKEsG-Mono parameter scan range was set to the following:

$$\rho[0.00714,\ 0.15,\ 0.25,\ 0.35,\ 0.45,\ 0.55,\ 0.65,\ 0.75,\ 0.85,\ 0.95],$$
$$\text{EFIELD}[\log(1 \cdots 9)\cdot 10^{(-2\cdots 2)}],\ \text{CMUL}[\log(1 \cdots 9)\cdot 10^{(-5\cdots 0)}]$$

Thus, for the DKEsG case, and taking into account the corresponding 30 DKEsG-Neo calculations, there are 25,630 independent BoTs composed by one task, 30 % of which have a duration that exceeds 5 hours. For the ISDEP case, 10^4 trajectories must be calculated per ρ value, resulting in a total of 500 BoTs (10^5 ISDEP tasks). Thus, task submissions should not be randomly performed because the maximum duration of pilot jobs is usually 24 or 48 h (depending on the wall-time of the remote queue where the pilot falls). Then, it is necessary to establish a mechanism to prevent the system from idle pilots when many long tasks are waiting and when they exceed the remaining pilot lifetime. This can be performed by prioritising longer executions over the rest, i.e., following a *longest job first* (LJF) policy that allows filling pilots with shorter tasks when longer tasks cannot be executed because of their duration. For this

experiment, a higher priority is set for every task that should be run first using the *DRMAA_GW_PRIORITY* parameter. Thus, three DRMAA applications have been executed at a time: one for prioritised DKEsG tasks calculating surfaces ρ[0.75, 0.85, 0.95] that last more than 5 hours and two other applications, which manage ISDEP and the remaining DKEsG tasks. All of them are configured with the same options used for the application executed in the initial test, but with exceptions. The former adds the prioritised requirement to their tasks and constrains the submissions to only pilots with more than 23 hours of life left. The other two applications only constrain the submission to pilots with more than 2 hours of life left. Those is done by setting accordingly the *QUEUE_MAXTIME* as additional requirement, as has been explained in Chapter 8 and Section 4.3.

To maintain the Provisioning policy established in the initial test, i.e., to only use 500 pilots from the 550 submitted, the $MAX\_RUNNING\_USER = 500$ option was set in the GridWay configuration of the Scheduler. Then, the three applications used in this work can queue their tasks in GridWay for scheduling without limitation, but only a sum up to 500 tasks from these three applications will be maintained as running on pilots. These will be the ones with the highest priority, as explained with Algorithm 3. Nevertheless, the maximum value previously configured will also limit the number of pilots to 500, not only the number of tasks. To solve this issue, two GWpilot Factories are configured using different UNIX accounts, but in shared mode and with the same user certificate. These Factories will manage a maximum of 250 and 300 pilots, respectively.

Results

The LJF policy hinders from showing a chart similar to Figure 9.1 due to GridWay overheads for short tasks are very high and they make it illegible. However, these high overheads are without importance for makespan because of their low priority. In contrast, the GridWay overheads for long tasks suffer smaller variations than the preliminary test due to their wall-time, that is, GridWay does not frequently schedule a high volume of this type of tasks. However, this overhead continues to be tied to the availability of pilots. On the other hand, the pilot and the application overheads are maintained constant at ∼41 s and ∼53 s during the calculation, as expected.

Therefore, the interest now is to measure the influence of failing pilots on the accomplishment progress of long tasks and especially of ended pilots on the wasted time by cancelled short tasks. The time spent on failing tasks can be considered another type of overhead now related to the overall computation and makespan. In this sense, the accumulated wasted time due to this reason is as long as 101 d 7 h 41 m. Additionally, to correctly evaluate this overhead (∼2.2 %), it is noteworthy to know that the accumulated time spent by tasks in execution was 13.14 years. A sequential execution on the E5-2667 machine would have lasted approximately 10.3 years. This indicates that most of the resources from *gisela* VO are closer to the performance of Xeon X5365 because the calculation would have lasted for ∼14.93 years on such a machine. Additionally, the final makespan of the complete test was 9 d 19 h 25 m, that is, ∼562 times faster than a sequential execution on the Xeon X5365 machine or ∼388 times faster than that on the Xeon E5-2667 machine.

9.6. Conclusions

In this chapter, the original scheduling algorithm of GridWay has been redesigned to better support the load excess from pilot processes. The result achieves the performance needed to allow users to implement advanced Workload scheduling techniques. Additionally, new methodologies to improve Provisioning have been devised. These features can be properly exploited in a multi-application or multi-user environment and even with multiple infrastructures or federations.

To demonstrate these facts, a real physics study that requires results from two legacy applications was performed. Thus, as a first conclusion, the new features implemented have demonstrated an impressive performance improvement in terms of makespan for the selected applications (DKEsG and ISDEP). However, the makespan reduction was not performed by simply increasing the throughput with a larger number of resources. GWpilot makes the most of the infrastructures by improving Provisioning and reducing overheads, as those directly influence the task turnaround. The measurements presented demonstrate this affirmation because these types of calculations are very representative, i.e., they are composed of short and long jobs managed at the same time. Additionally, the Application layer better profits from the resources by implementing their own policies, in this case, long jobs are prioritised in a shared environment. Therefore, the scalability of GWpilot and its suitability for other types of codes was guaranteed.

The features and experiments presented constitute a first step to customise the scheduling with GWpilot. The combination of ranking (in Task Scheduling) and overloading the infrastructure with unused pilots improve the appropriation of resources through the time, but with no communication between both layers. To describe how Workload Scheduling and Provisioning can actually work together, the tools available for characterisation have to be extensively explained. Additionally, the study of turnaround should originate such a formalism that allows other systems to stack over GWpilot. These aspects are achieved in Chapters 10 and 11.

As an additional result of this work, all the necessary tools for performing an efficient and extensive parameter scan of DKEsG are ready to be placed into production. ISDEP also counts now, with a standardised management for distributed platforms. This is because GWpilot avoids the performance and compatibility limitations of other techniques previously used for these applications. Furthermore, new interesting results related to the Neoclassical ordering violation in fusion plasmas have been obtained by comparing the results provided by both codes (see Subsection C.4.2 in the Appendix C).

Chapter 10

Customising the Whole Scheduling at User-level

10.1. Introduction

The features listed in Section 7.2 accomplish a set of requirements (see Section 3.2) that other systems do not. The main benefit of the tools presented is that they can be combined to build a broad range of scheduling algorithms. In particular, the main added advantage of GWpilot over other pilot systems is the capacity offered to users and developers to dynamically build their own scheduling policies, but assisted by the framework. With GWpilot, the scheduling is actually simplified into the three-level hierarchy logically established in Subsection 2.3.1: the User or Application layer, where the applications dynamically perform their Workload Division and Job Building; the Task Scheduling layer, where the pilot system performs the task-pilot matchmaking following the user-defined requirements; and, the Provisioning layer, where the system search in the federations for resources that fit those requirements. Thus, one main objective of GWpilot is to facilitate the exercise of dynamically propagating down the requirements of any application from the user-level to the lower layers.

For this purpose two new mechanisms are needed, which are not shown through previous chapters and are also required for Multilevel Scheduling: the capacity of fully characterisation (and monitoring) of resources, and the communication among layers. However, users are not alone. Communities demand to improve specific aspects such as the global throughput or the fair-share. The generic configuration of Workload and Provisioning layers supports their requirements, but also implies a new world of possibilities.

10.2. Dynamic and customisable characterisation

The level of scheduling customisation of any pilot system depends on the language used to characterise resources and tasks running, as well as on how the users can establish preferences or constraints for the execution of their tasks based on this language.

Every member of the Host Pool is described by means of unstructured label-value pairs. With the exception of the identifier of every member, they can operate

as un-typed variables, i.e. they can dynamically take a numerical value or turn into an arbitrary string. Additionally, the number of tags is not limited. Therefore, the characterisation of any resource from any DCI can be fully stored. GWpilot also performs this action with the information supplied by the enrolled pilots (see Table 7.1).

In this sense, the framework actually gives the possibility to users to directly customise some tags. The mechanism does not imply the modification of pilot implementation by the user. Tasks running can communicate with pilot through a named pipe, the path of which is unique for each task and it is stored in an environment variable (\${*PILOT_PIPE*}). Subsequently, tasks can write on that pipe customised tags declared as *PILOT_*\${*GW_USER*}*_VAR_* $<$ *number* $>=<$ *arbitrary value* $>$. These tags will be published in the Host Pool and are maintained among task executions. Thus, the user only has to submit tasks which push the needed information (an integer, float or string) into the supplied path in order to specify that any other intermediate file was stored in a scratch directory, any specific profile has been performed or any configuration has been done at the remote resource.

Any user can select multiple tags from resource descriptions to formulate a Boolean expression as a requirement, which can be combined with a numerical expression to rank every candidate host (those pilots for which the requirement expression is true). In addition, users can include the supplementary tags dynamically customised as the Scheduler fully supports the resolution of these expressions. Thus, those pilots with higher ranks are first used to execute his tasks (and to submit his pilots, as will be explained in Subsection 10.4).

Therefore, as a result of the integration of GWpilot framework, users and developers count on tools to really build a personal scheduling. Particularly, they can:

- dynamically inspect and filter those tags through the *gwhost* command;
- notify any custom characteristic from their running tasks inside pilots;
- dynamically establish requirements and ranking expression based on these tags in their task descriptions to fit their computational needs.

However, the GWpilot also offers some other useful tools that allow users to dynamically know:

- what tasks are running in certain pilots (through *gwps* command) and how many tasks were successfully executed or failed in these pilots, as well as their accumulated transfer and execution times (through the *gwacct* command);
- what pilots have been submitted to real sites and which are really running in them (also with *gwps*). Same accounting is performed with pilots, so users can know the same accumulated registers than the ones provided for tasks (with *gwacct*).

These tools facilitate the tuning of GWpilot configuration for a specific calculation. They also allow users to select pilots that are effectively accomplishing tasks faster than others. Additionally, wrapper and middleware logs are stored in a simple tree structure based on the task and pilot numerical identifier, which facilitates the troubleshooting.

10.3. Feasible Workload Scheduling

Through last section the simple tag-based language and the tools to personalise requirements have been introduced. However, the reader only get a glimpse of the possibilities those mechanisms offer.

First, the meaning and classification of the tags to be notified to the PiS and, consequently, published into the Host Pool deserve a more complete explanation because they enable more advanced scheduling mechanisms than those that are associated with a typical characterisation of the resource. In this sense, in addition to the generic static and dynamic characteristics of a resource, such as the CPU type or the CPU consumption, pilots can specifically notify specific tags. For example, the real memory and CPU usage of the running tasks allow the Scheduler to properly detect performance losses and consequently to initiate the checkpoint of task and subsequently its migration.

Moreover, GLUE-style tags can be notified not only to describe the configured middleware (such as the *close-SE* available) but also to describe a virtual queue description that could be customised to provide the Scheduler with a number of fictitious or virtual free slots and the distinguished names of allowed users. By doing so, not only the multi-task and the multi-user (MUPJ) capabilities are enabled, but also resources composed by multiple cores or GPUs can be managed by the Scheduler. Additionally, the remaining wall time in the remote resource (i.e. how many time remains for the end of pilot) is included in this virtual queue. Thus, Scheduler can fill pilots with tasks of adequate duration.

Furthermore, some files related to a task, i.e. executables, inputs, and outputs, are declared with their MD5 code to be cached for later reutilisation by other tasks. Any tag declared is directly visible by either the user or the developer, who could include it as a requirement to build complex workflows based on file dependences, which will be fully supported by the Scheduler.

The pilot implementation goes beyond offering the default tags contained in Table 7.1 to the user, which are feasible for a broad range of calculation types, but they could not be used to accomplish some concrete problems. The proposed mechanism to customise the characterisation of pilots facilitates the software deployment to VO administrators or even temporarily to unprivileged users. This characterisation methodology also allows the inclusion of advanced scheduling algorithms belonging to the Application layer, especially those usually provided by third-party self-schedulers, which can now be added to the system without re-implementing the functionality of a resource broker. The provided possibility of direct profiling and monitoring on the WN (properly estimating outbound bandwidth or the application performance, for example) is essential for improving and reactivating the task chunking. Additionally, advertising the availability of some files facilitates the data-allocation policies. Moreover, an effective virtual type of advanced resource reservation, based on pre-emption, could be allowed in long-term GWpilot systems if this mechanism of file awareness is combined with the checkpointing features provided. This can be very useful for supporting some complex scheduling algorithms [32], MPI, and very long executions, or enabling on-demand prioritisation of some calculations over others, such as real-time applications.

10.4. User-guided Provisioning

The capacity of the GW PiF to inspect the requirement and rank expressions of tasks enables such type of user-guided Provisioning. These preferences are dynamically taken in consideration when a pilot is submitted to progressively improve the quality of appropriated resources. The difference with approaches such as [215], is that this mechanism is directly managed and customised by any user without requiring the modification of the pilot system code. In addition, it is more generic than other approaches [204] because does not constraint the submission of one pilot to accomplish a concrete task. Moreover, the guidance in Provisioning is completely refined because the PiF is able to distinguish among the customised characteristics of pilots to properly select resources.

The requirement expression of every pilot is simply built with the logical disjunction of the requirements of every task. However, the following formula builds the rank expression for user-guided pilots:

$RANK = \sum p_i \cdot RK_i \cdot n_i/n$

where n is the total amount of tasks in the system, n_i is the number of identical tasks with the same (RK_i) rank expression and p_i is their current priority given by the Scheduler. Thus, the GW PiF maintains the fair-share among running applications. Moreover, as the PiF can be configured to deal with several users, it allows such a type of user fair-share in Provisioning. Additionally, the number of guided pilots can be limited to a percentage. This mechanism provides great results and has been used in the tests shown in Subsection 10.6.

10.5. Effects of configuration on the scheduling layers

Despite of the advantages provided to the users with dynamic scheduling policies, configuration parameters are not suitable for being dynamically modified. This is justified by the need of control by an administrator when GWpilot is used by multiple users and by the request of communities to improve certain metric, such as the throughput or the resource utilisation.

Some of these parameters are specific to the PiS or have been mentioned as pilot parameters. For scheduling, the pilot pulling interval, the number of tries and the maximum number of pilots are more important. These options have been already explained through Chapters 8 and 9. In addition to them, requirement and rank expressions are allowed as configuration option of the PiF. However, it could be desirable to partially perform the Provisioning guidance in some types of calculations. For this reason, the number of guided pilots can be limited to a percentage in the configuration. Another powerful feature is its capacity to perform a flooding of the infrastructures with a limited number of pilots, but above the real need. Therefore, administrators can effectively control the pilot production of the system.

However, as GWpilot is an embedded system, the options inherited from GridWay must be also included in this category because they have obvious implications on the GWpilot behaviour. Thus, the configuration options described in Table 2.1 take new meanings that must be clarified.

For example, the maximum number of hosts limits the volume of resources (i.e.

Table 10.1: Static options for GWpilot configuration.

Scheduler loop options:	
$SCHEDULING\_INTERVAL$	Interval to perform a new scheduling of pending tasks and pilots
$DISPATCH\_CHUNK$	Maximum number of tasks and pilots dispatched in every scheduling interval
$MAX\_RUNNING\_RESOURCE$	Maximum number of pilots concurrently submitted to same site (or task to same pilot)
$MAX\_RUNNING\_USER$	Maximum number of simultaneous running task and pilots per user
Dispatch priority of a pilot or task (j): $P_j = \sum_i w_i \cdot p_{ij}$**, where w is the weight and p the priority contribution of every** $i = \{FP, SH, WT, DL\}$	
FP_USER,FP_GROUP	Fixed priority per user or group (default 0)
SH_USER,SH_GROUP	Ratio of submissions of a user or group over the rest (default 1)
SH_WINDOW_SIZE	Timeframe over which user submissions are evaluated (in days)
SH_WINDOW_DEPTH	Numer of frames (present frames are most relevant)
DL_HALF	When pilot or task should get half of the maximum priority assigned by this policy (in days)
Suitable priority of a resource (h): $P_h = f \cdot \sum_i w_i \cdot p_{ih}$**, where w is the weight and p the priority contribution of every** $i = \{RP, RA\}$**. f is** 1 **when resource h is not banned.** (Note that UG policies should be disabled)	
RP_HOST, RP_IM	Fixed priority per site or per every resource discovered by an IM
FR_MAX_BANNED	$T_\infty \cdot (1 - e^{\Delta t / C})$, where T_∞ is the maximum time that a resource can be banned, Δt is the time since last failure, and C is a constant that determines how fast the T_∞ limit is reached
FR_BANNED_C	The value of the C constant in the above equation
GWpilot parameters:	
-i $< PI >$, -t $< T >$	Pilot pulling interval and number of tries
-n $< unsig.\ int >$	Maximum number of pilots
-w $< unsig.\ int >$	Maximum waiting time in grid queues or for VM creation
-o $< unsig.\ int >$	Over-submission of pilots
-c $<REQUIREMENT>$	Expression to constraint pilot submission
-r $<RANK>$	Expression to rank resources for pilot submission
-g $< unsig.\ short >$	% of guided pilots
-nosec, -s	Unsecure mode and shared mode of PiS

sites and pilots) that the system can manage. Other options limit the number of clients (application calls), users, tasks or pilot jobs in the system. Those options with a special significance were summarised in Table 10.1. In this sense, fair-share policies, i.e. fixed (FP), share (SH), deadline (DL) and waiting-time (WT) policies, could negatively impact on the effective dispatching time of certain pilots and tasks. In addition, resource (RP) policies constraint the use of resources and are mainly related to Provisioning. In any case, they allow the management of the system in a way more close to a PMS service [17, 202].

Many of the fair-share policies will be disabled if pilots cannot be shared among different users. For this reason, PiS and PiF sharing were allowed to enable MUPJs. Running in this mode, if the PiS is owned and initialised for a specific user, for exam-

ple, the production manager of a VO, the other users must allow its distinguished name (DN) into the local *grid-map* file to correctly stage-in and stage-out their files through the GridFTP. Nevertheless, although this procedure is feasible, it is an insecure method that cannot accommodate the policies of some infrastructures [261] because the pilots do not isolate users who are executing codes with the same DN role of the pilot owner. This mechanism is allowed under the responsibility of the administrator or the VO manager, and because the user-identity-switching tools (e.g., gLExec [186]) are not widely installed and they are currently only required by large production VOs.

Other parameters are fundamental for both Task Scheduling and Provisioning. For example, the weight of rank (RA) expression in the task or pilot jobs should be enabled (i.e. set to one). Nevertheless, the usage (UG) statistics prioritise resources with shorter execution times per job; consequently, they are counter-productive for pilots and should be disabled. The failure rate (FR) policies allow users and administrators to automatically discard pilots and sites that accumulate persistent task failures. The dispatch chunk and the scheduling interval determinate the volume of the tasks and pilots dispatched, that is, the productivity. However, pilot submission should be lesser frequent than task generation. Subsequently, these last parameters and the pilot pulling interval mainly determinate the task turnaround overhead as has been demonstrated in Chapter 9.

10.6. Experimental demonstration

Through the experiments performed in previous chapters the performance of GWpilot was compared with other two pilot frameworks as well as a set of distinguishing features have been demonstrated, such as: its capacity of provisioning resources in cloud federations, its compatibility with legacy applications or its support for implementing more advanced scheduling policies.

However, those tests only provide with a measurement of the performance gain of GWpilot and several advantages with respect to other systems, but they do not show how users or developers can dynamically personalise and guide whole scheduling. That is, users should be able to customise the characterisation of resources and subsequently to direct the task distribution as well as the Provisioning. Therefore the objective of the experiments performed in this section is to demonstrate that these actions can be straightforwardly performed, without modifying either the code of GWpilot or the applications.

10.6.1. Proposed calculation

The interest is to perform real calculations as any user usually does, i.e. customising requirements of his legacy applications and starting several calculations at the same time. The approach is to remark the importance of scheduling variable short tasks (with an execution time shorter than 20 minutes) in an overloaded infrastructure because this is the worst potential scenario that a pilot system (and any scheduler) can face. Thus, the features introduced should demonstrate that they improve the execution of user applications while not overloading the GWpilot system with excessive overheads. This will be demonstrated through this section executing again the long multiplication used in Chapter 7, but also several instances of a legacy application such as DKEsG at the same time, simulating a multi-application and multi-user

environment.

A real example of calculation with DKEsG-Mono is the determination of the effective ripple in a fusion device (see Section C.3 for further information). For this purpose, 24 variations of the standard configuration of the TJ-II [260] device, 72 variations of the plasma collisionality, and 140 radius indexes have been selected, while the electrical field is considered constant. Therefore, this input parameter combination results in $24 \cdot 72 \cdot 140 = 241,920$ tasks.

In this case, every independent DKEsG-Mono instance is a short-duration task whose CPU consumption is directly proportional to only one parameter, the plasma radius index (i.e. the toroidal flux, ρ). Therefore, DKEsG-Mono is a common parameter sweep application with a controlled CPU time variability that can be used to demonstrate the feasibility of the mechanism to customise scheduling at user-level. This is so due to it allows a specialised characterisation based on the execution profiling.

Moreover, to show an effect similar to that found in a competitive multi-application environment (but also controlled for further analysis), the calculation of the effective ripple is split in three parts to be carried out by three different DKEsG instances. Tasks with a lower weight are interspersed along the test. For this purpose, tasks are ordered for later individual submission in a numerical sequence of radius as:

```
Slice 1: (2,47,92;3,48,93;... 16,61,108) + (139,140,141)
Slice 2: (17,62,107;18,62,108;... 31,76,121) + (137,138)
Slice 3: (32,77,122;33,78,123;... 46,91,136)
```

Furthermore, tests are mainly composed by the DKEsG-Mono tasks needed for the calculation of the effective ripple, but long multiplication is also executed at the same time. The purpose is to use long multiplication as a competitive application in the system and to subsequently analyse its interference on scheduling, because it has different computational requirements.

Naturally, those three DKEsG applications will be started at the same time as the long multiplication. Additionally, the radius order assures that the CPU time requirement is increased along the experiment. Every application (long multiplication included) is limited to 250 tasks, and then the maximum number of tasks in the system is 1,000.

Finally, estimating the duration of both executions is necessary. If calculation of the ripple were performed sequentially on the first selected machine described in Table 10.2, it would take approximately 3 years and 45 days, while it would last 36 hours and 30 minutes if 750 cores of that machine were used. On the other hand, long multiplication on 250 cores would take approximately 32 hours if the input set that assures tasks above 10 minutes ($< lower\ limit >= 9$) were selected.

10.6.2. Customising characterisation and scheduling

Profiling the calculation

To introduce any accurate scheduling policy, DKEsG-Mono execution must be profiled. Unlike long multiplication, which is entirely based on integer arithmetic, DKEsG-Mono execution time depends also on floating point performance, cache size and other hardware parameters. Thus, latest processors should compute DKEsG-Mono faster. This issue, far from being a drawback, will be used in this experiment to show how different policies can be enabled for several applications at the same time.

Table 10.2: Average DKEsG-Mono execution times obtained from reference machines for some indexed radius in the standard TJ-II configuration. The updated release of DKES [245, 248] is used.

TJ-II index for radius (cm)	TJ-II normalised toroidal flux ρ (0-1)	Xeon X5365 3 GHz (Launch date: 2007)	Xeon E5620 2.4 GHz (Launch date: 2012)
2	0.00714	58.10 s	48.33 s
25	0.171	179.54 s	150.21 s
48	0.336	297.33 s	247.80 s
71	0.500	348.14 s	291.28 s
94	0.664	523.58 s	439.59 s
117	0.829	624.92 s	524.07 s
141	1	751.05 s	631.95 s

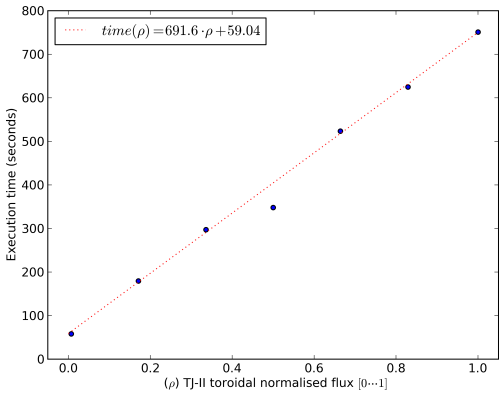


Figure 10.1: Linear fitting of the values in Table 10.2 for the he X5365 processor. This is the suggested profile of DKEsG-Mono on that processor.

These applications will be concurrently managed by GWpilot, resulting in performance improvements for all of them.

In the case of the calculation proposed, DKEsG-Mono execution time ranges from 58 to 751 seconds on a Xeon X5365 3 GHz, but it is lower on a Xeon E5620 2.4 GHz (see the third and fourth columns in Table 10.2). As shown in Fig. 10.1, these data can be fitted into a polynomial function. Then, the speedup obtained running DKEsG-Mono on any other processor can be expressed by drawing a parallel line above (less performance) or below (better performance) that function. Additionally, the speedup for every normalised flux (ρ) can be calculated with the following formula:

$$speedup(\rho) = \frac{<wall\ time>}{59} + \rho \cdot 692$$

Characterisation allows scheduling guidance

Now, the objective is to perform a simple customization of the GWpilot scheduling capability based on the personal profiling of the application previously performed. This purpose motivates the election of a real application such as DKEsG. As commented through previous chapters, advanced policies can be dynamically included in GWpilot by simply including specific pilot tags in the rank expression. For example, if

the long multiplication is used in this experiment, only including a ranking expression based on the CPU speed in templates will be enough to obtain a valuable performance improvement:

$$RANK = PILOT\_CPU\_MHZ$$

However, the intention is to go beyond this, showing how users and developers can introduce their personal tags since they are who really know the behaviour of their applications.

Therefore, the $PILOT\_\$\{GW\_USER\}\_VAR\_1$ tag will publish the profiling established for DKEsG-Mono, i.e. the *speedup* obtained by every execution. This also monitors the performance of the appropriated resource, allowing the reactiveness against performance losses due to overloads or overbooking. However, the $PILOT\_\$\{GW\_USER\}\_VAR\_1$ has to be filled by the task whenever it ends its execution. For this purpose, a script that wraps DKEsG-Mono execution at WNs will calculate *speedup* and publish it with this statement:

```
echo "PILOT_${GW_USER}_VAR_1 = ${speedup}" > ${PILOT_PIPE}
```

On the other hand, the following DRMAA statements must be introduced into the code section of task creation: the former to set a ranking policy based on the customised tag and the latter to indicate that such a task has to be scheduled to a pilot:

```
drmaa_set_vector_attribute(<task description id>,
     DRMAA_GW_RANK, 'PILOT_${GW_USER}_VAR_1')

drmaa_set_vector_attribute(<task description id>,
     DRMAA_GW_REQUIREMENT,'LRMS_NAME="jobmanager-pilot"')
```

Thus, any unskilled developer can perform these actions, and does not need to modify any code of the GWpilot system, including the pilot itself.

Moreover, these modifications are only shown as an illustration of the usual procedure with legacy applications. DKEsG is adapted following the producer-consumer library described in Section 4.3 and consequently, these requirement and ranking expressions actually are arguments of the application.

10.6.3. Competitive tests

The benefit of guiding Provisioning and Task Scheduling together, and not simply guiding the latter, is the key achievement that this experiment is trying to demonstrate.

For this purpose, three virtual machines with GWpilot are booted. These instances are configured with the static options for GWpilot enumerated in Table 7.2. This assures the comparison with the other systems. Moreover, for the sake of comparison with the experiments performed in Section 7.4, a GWpilot instance must act as a control test, i.e. supporting DKEsG and long multiplication without the aforementioned ranking and without enabling the GW PiF Provisioning guidance. On the other hand, custom characterisation and ranking policies are enabled by the user applications running on the other two virtual machines. Finally, Provisioning guidance is also configured in one of these two. With this differentiation, the performance gain

obtained by each scheduling technique can be measured. Thus, those three deployments are called along this experiment as control test (*ct*), standalone ranking (*sr*), and guided Provisioning (*gp*).

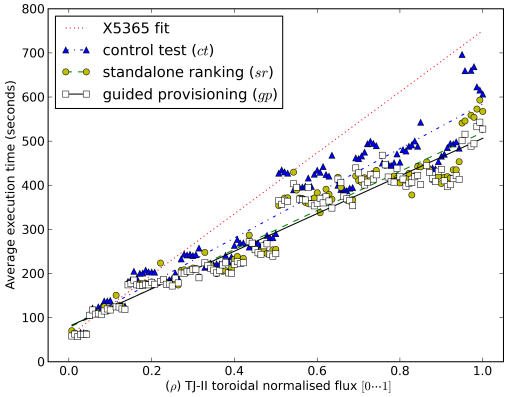
All tests will rely on the IS to find free resources to immediately distribute pilots. However, the user-ranked approach enabled in *sr* and *gp* differentiates the Resource Provisioning from the Task Scheduling. Both approaches expect more WNs to be obtained, the best ones of which will be retained if pilots wait a certain time in low overloaded queues. It was commented through the experiment in Section 7.4 that application instances (tasks) in the system usually do not reach the maximum number of pilots that are available in a determined time. This behaviour is especially noticeable with DKEsG, because it relies on the producer-consumer library described in Section 4.3. Then, the execution state of their tasks is sequentially checked in the polling interval, in this case, 0.1 s. However, GWpilot takes advantage of this underutilisation when workload policies are enabled as in experiments in Section 9.5, although with less efficiency because the over-submission of pilots are not enabled now. That is, GWpilot achieves an improvement in the quality of resources as a function of time: the best pilots will be selected to run a task and the rest will die when no work is assigned to them.

Besides, *gp* tests finally connect user policies to Provisioning. The PiF will include the preferences defined in the task descriptions when pilots are submitted (see Section 10.4). This scheduling mechanism considers that there is no characteristic that the IS could provide, including the published number of free slots from every resource. That is so because the IS does not assure that these slots are actually available for a specific VO user. However, the GW PiF can safely use the characterisation of pilots to improve future Provisioning, even if this characterisation was customised by the user. Additionally, the PiF will use the ranking sentences included in the task descriptions, but in a limited way. The PiF is configured to only guide half of pilot submissions. That is so to allow GWpilot to better explore the whole infrastructure in order to search for more suitable resources, but assuring the suitability of resources in a 50 % at least. In addition, it is of interest that the influence of these ranking statements on the scheduling mechanism was unbalanced. The intention is to create an experiment close to reality, where there will always be more tasks of some type than other. Additionally the tests have been performed three times to assure accurate results.

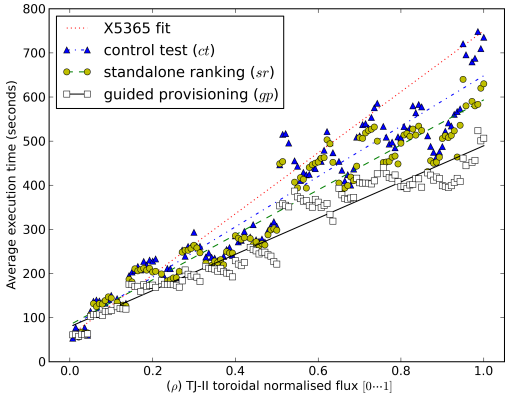
10.6.4. Results

The time spent by the tests is a good introduction to the advantage offered by GWpilot over other solutions, because control test (*ct*) relies on the same configuration parameters used in the experiments described in Section 7.4. Now, the experiments are composed of several applications with variable completion times. Additionally, different user policies are applied on long multiplication and on DKEsG executions. Consequently, differences between the *ct* and guided Provisioning (*gp*) tests range from 56 minutes (Test 1) to 8 hours (Test 2). Table 10.4 shows the makespan of each application and their accumulated spent time.

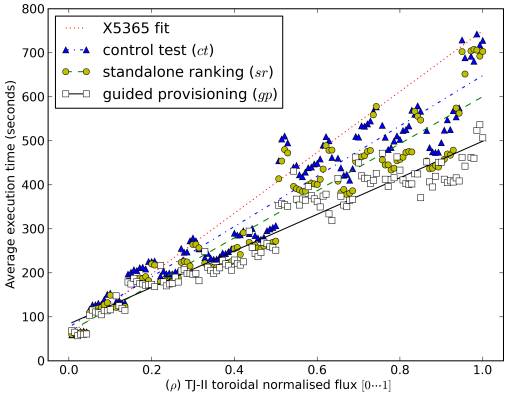
According to these data, an important reduction (from 11 % to 20 %) of accumulated spent time in the *ct* tests is achieved by the *gp* tests. However, the standalone ranking (*sr*) tests do not achieve an averaged improvement as it could be expected. This behaviour demonstrates one of the motivations of this thesis, i.e. it is necessary



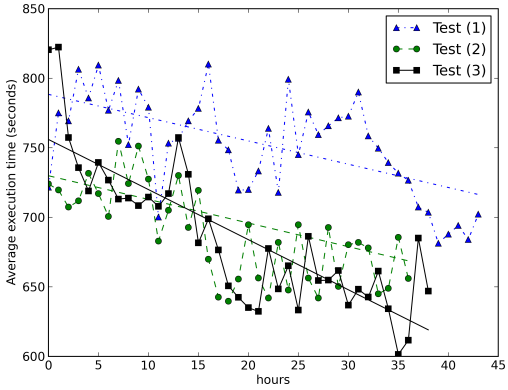
(a) Test 1. Execution time average of DKEsG-Mono per normalised flux (ρ).



(b) Test 2. Execution time average of DKEsG-Mono per normalised flux (ρ).



(c) Test 3. Execution time average of DKEsG-Mono per normalised flux (ρ).



(d) Task execution time average per hour (in seconds) of long multiplication in the three guided Provisioning (gp) tests (with $n = 90,000$).

Figure 10.2: Average times obtained in the competitive tests that demonstrate the improvement through the time with guided approaches.

Table 10.3: Results obtained in the competitive tests that evaluate the renovation of pilots and performance. Values for *tasks done per minute*, *pilots discarded per minute* and the *number of calls per second of every pilot* are measured after the system had enrolled at least 800 pilots and before the last 1,000 tasks were remaining to be representative of the usual behaviour of GWpilot.

Test	**Type**	**Tasks**			**Pilots**			
		failed	done/m	max. done/m	submitted	enrolled	discarded/m	calls/s
(1)	*ct*	5,140	104.41	241	30,882	9,107	2.95	62.09
	sr	5,307	122.03	271	24,844	5,973	1.96	64.78
	gp	4,625	124.96	268	22,631	6,049	2.05	67.05
(2)	*ct*	6,322	100.05	238	27,421	8,134	2.59	65.76
	sr	4,962	121.30	254	26,041	7,080	2.53	67.31
	gp	8,663	132.72	294	27,631	9,049	3.01	66.84
(3)	*ct*	4,718	89.68	216	21,572	6,227	2.36	59.20
	sr	8,726	91.82	231	13,295	6,698	2.37	61.26
	gp	3,954	126.05	276	15,497	7,055	2.88	68.05

Table 10.4: Makespan values obtained through the competitive tests that compare guided and not guided scheduling.

Test	**Type**	**Makespan**				
		long multiplication	DKEsG-Mono Slice 1	DKEsG-Mono Slice 2	DKEsG-Mono Slice 3	accumulated
(1)	*ct*	1 d 20 h 54 m 23 s	1 d 12 h 09 m 07 s	1 d 19 h 16 m 32 s	1 d 20 h 23 m 50 s	7 d 00 h 43 m 52 s
	sr	1 d 21 h 29 m 12 s	1 d 09 h 41 m 16 s	1 d 12 h 11 m 35 s	1 d 12 h 59 m 43 s	6 d 08 h 21 m 46 s
	gp	1 d 19 h 58 m 07 s	1 d 08 h 36 m 55 s	1 d 11 h 46 m 01 s	1 d 13 h 49 m 22 s	6 d 06 h 10 m 25 s
(2)	*ct*	1 d 21 h 56 m 13 s	1 d 19 h 50 m 51 s	1 d 21 h 07 m 36 s	1 d 21 h 43 m 34 s	7 d 12 h 38 m 14 s
	sr	1 d 18 h 08 m 21 s	1 d 17 h 32 m 11 s	1 d 18 h 54 m 07 s	1 d 16 h 40 m 27 s	6 d 23 h 15 m 06 s
	gp	1 d 13 h 51 m 20 s	1 d 09 h 18 m 16 s	1 d 10 h 56 m 42 s	1 d 12 h 01 m 03 s	5 d 22 h 07 m 21 s
(3)	*ct*	1 d 21 h 34 m 02 s	1 d 18 h 23 m 14 s	1 d 20 h 56 m 40 s	1 d 21 h 29 m 42 s	7 d 10 h 23 m 38 s
	sr	1 d 20 h 23 m 52 s	1 d 17 h 47 m 48 s	1 d 20 h 08 m 49 s	1 d 20 h 03 m 18 s	7 d 06 h 23 m 47 s
	gp	1 d 15 h 49 m 48 s	1 d 08 h 17 m 00 s	1 d 10 h 51 m 06 s	1 d 11 h 46 m 44 s	5 d 22 h 44 m 38 s

the guidance of Provisioning because resources finally provided by remote sites could not fit the needs of the applications or they could even arbitrarily fail. The GW PiF in the *ct* and *sr* tests does not perform any selection of resources where pilots will be submitted with exception of avoiding busy and banned sites. Therefore, when a pilot is discarded, it is arbitrary replaced by a new one. Even, this new pilot can be in the same WN than the replaced one if the holding site was not banned. Therefore, the chance to obtain improved resources is based on multiple attempts, but they are better retained in the *sr* tests. This circumstance is especially evident in Test 1, where resources with high speedups profiled are appropriated at the beginning of the calculation. Paradoxically, the complete Test 1-(*sr*) was slower than Test 1-(*ct*). That is because pilots more suitable for long multiplication executions (higher CPU speeds) are less retained if they do not entail an improved profile for DKEsG-Mono application.

In any case, the algorithm presented in this study devoted to firstly utilise the more powerful pilots in the pool has resulted in a continuous improvement of the resources offered to the user in a ratio proportional to the discarding rate. This fact is supported by the number of tasks per minute considered *done* by the applications, which is increased from *ct* to *gp*. Naturally, values for both rates written in Table 10.3 were registered when the system maintained the four application running to assure accurateness.

To illustrate how the continuous improvement of resources impacts on the accomplishment of tasks, averaged execution times are shown in Figures 10.2. However, to differentiate the influence on every application, two types of graphs are included. First, the execution time average of DKEsG-Mono tasks per normalised flux is depicted in Figures 10.2-(a, b, c). To compare the speedup obtained, the data belonging to every test are fitted into a polynomial function. Additionally, the line representing the performance of the reference machine is drawn. A supplementary fourth Figure 10.2-(d) shows the evolution of the execution times of long multiplication for $n = 90,000$ through the *gp* tests. This latter graph is included only to demonstrate that, despite of pilots are preferably submitted to sites with resources suitable for DKEsG, the ranking approach in Provisioning also assures an improvement for long multiplication. Thus, this approach achieves a progressive reduction of the average CPU consumed by their consecutively submitted tasks. These conclusions are supported by the makespan obtained by every application and compiled in Table 10.4. In particular, the calculation time of every DKEsG slice is reduced from 11.2 % to 25 % and the long multiplication from 3.4 % to 13.4 % in *gp* tests.

Nevertheless, approximately 25 % of the discarded pilots is consequence of grid job errors, so the improvement translated to a reduction in the makespan is smaller. Additionally, the number of failed tasks in the pilots' execution deserves a deeper explanation. Although GWpilot is more stable than the traditional early-binding methods, it cannot avoid errors produced by the middleware, network cuts or any other typical problem related to the remote execution of pilots. This is the case for the failed executions shown in Table 10.3, which are primarily the consequence of a pilot dying while a task was running in it or a task being dispatched to a dead pilot that has not been discarded yet. However, they represent less than 4 % of the total submitted executions in every test, while the aforementioned percentage of failed pilots rises to 75 % (note that many pilot jobs were discarded before the 30-minute threshold set in the PiF configuration).

10.7. Conclusions

In this chapter, the needed techniques to allow customising scheduling at user-level have been presented. They consist in the combination of the user's capacity to fully characterise pilots and to guide both Task Scheduling and Provisioning. Moreover, this feature is complemented by the configuration options that allow administrators to preserve fair-share and global throughput as well as improve other global aspects. Furthermore, the approach enables developers to build a wide range of advanced Workload Scheduling techniques that are completely unfeasible for other pilot systems.

To demonstrate the benefits of the communication between Application, Task Scheduling, and Provisioning layers, three competitive experiments were performed in a multi-application environment. One performs a customised characterisation and also guides the provisioning, other only performs the characterisation, and the latter none. Results show that characterisation improves the calculation through the time, but the performance gap is achieved with a guided Provisioning.

The proposed mechanisms and abstractions described through this chapter are adequate to simplify the scheduling from the user's, developer's and administrator's points of views. It is noteworthy that they constitute a step forward on the achievement of a platform that fully supports the Multilevel Scheduling. However, although developers count now on tools to perform a customised scheduling, a formalism to adapt advanced scheduling or even to stack self-scheduling tools represent a clear asset. How this feature has been designed is addressed in the following Chapter 11.

Chapter 11

Modelling and Stacking Scheduling Tools

11.1. Introduction

The makespan of any application depends on the turnaround time of every task, as was explained in Subsection 2.3.4. Turnaround is very sensible with regard to overheads when short-duration tasks are scheduled. In contrast, long-duration tasks suffer from more failures that waste computational time. In general, developers of Workload Scheduling algorithms should estimate the time required by stage-in and -out operations but not the overheads produced by the pilot system. However, due to the performance stability of GWpilot, these overheads can be known *a priori* because they coincide with the intervals used by pilots for their operations against the GW PiS and with the time used by GridWay to dispatch tasks. For this reason, the maximum number of pilots, the pulling interval time against the GW PiS, the number of retries for getting a task or updating its state and the scheduling interval can be fully configured by the user. Additionally, the suitability of these values can only be obtained due to the troubleshooting mechanisms that GWpilot provides. The whole execution process can be followed by the user by either reading logs or typing commands. Users can use the compiled data of previous executions through the supported accounting. Therefore, these tools allow the fine tuning of the GWpilot configuration and the applications.

Furthermore, the capacity of completely characterise the overhead is a key feature to formulate a trustworthy model for task turnaround. This model jointly to the GWpilot support for Task Scheduling allow stacking specialised scheduling tools to GWpilot such as self-schedulers which can properly now run on grid and cloud infrastructures.

In this chapter a simplified task turnaround model is presented. Its suitability will be demonstrated through the statistical analysis of the executions performed in other chapters. Additionally, a methodology to incorporate external scheduling is explained. Whole suitability will be proved staking the self-scheduler framework developed in Chapter 6.

11.2. Modelling task turnaround with GWpilot

11.2.1. Simple turnaround model

The real turnaround time of each completed task is defined as the time difference between the moment when the task is queued in the UTQ (the GridWay Job Pool) for being dispatched and the moment when the system notifies that it is completed. This value represents the CPU time consumed by the application when it is executed on remote resources plus the transfer times and the overhead introduced by the middleware in Task Scheduling and Provisioning.

Unlike other pilot systems, related middleware overheads can be known *a priori* with the GWpilot configuration. Thus, when no failures are produced and enough amounts of pilots are available, the idealised turnaround (τ) is:

$$T = T_{sched} + T_{xfer} + T_{exec} \simeq t_{si/2} + t_{pi} + t_x + t_e = \tau \tag{11.1}$$

t_e and t_x maintains the same significance than T_{exec} and T_{xfer} in Equation 2.1. T_{exec} depends on the power of the resource selected, while T_{xfer} principally depends on the network and the amount of data transferred. Thus t_e as well as t_x should be estimated by a self-scheduler, but now GWpilot provides the characterisation of every pilot for this purpose. This is, the bandwidth, latency and power are published for every pilot. In addition, the tools needed to publish any specialised benchmarking of these or other performance aspects (for example, the disk throughput) are available in GWpilot.

On the other hand, the T_{sched} is the overhead related to the Task Scheduling mechanisms. It is split into two types of process ($t_{si/2} + t_{pi}$), with different modelling behaviour. First, t_{pi} is the overhead related to pilot notifications and the capacity of the GWpilot Server to process them. When no failures are registered, t_{pi} coincides with the pilot interval (PI) set in the GWpilot configuration.

$t_{si/2}$ stands for the time needed to obtain a suitable pilot to execute the task. GridWay Scheduler will prioritise some tasks over others and then will dispatch these tasks to pilots that accomplish their requirements (see Chapter 9). When tasks have identical requirements and enough amounts of pilots are already appropriated by the pilot system, $t_{si/2}$ only depends on the elapsed time between schedules set in the GWpilot configuration (the half of $SCHEDULING\_INTERVAL$).

Several algorithms can be improved only with this definition. However, this idealisation is far away from reality and it can be counterproductive for certain calculations as commented in Subsection 2.3.4.

To complete the simplified task turnaround without failures, the dispatching time has to be complemented with the availability of the resources. $t_{pi} + t_{si/2}$ always is wasted in the process although there would be enough pilots for a number of tasks ($\eta(t)$) in the system (running or waiting) at certain time (t). However, a task must wait to be scheduled when the (expected) number of pilots ($\xi(t)$) is lower than $\eta(t)$. Then, the additional dispatching time when the task is included in the pilot system is:

$$\Upsilon[t, t_{si/2}] = \begin{cases} \xi(t) >= \eta(t) & : 0 \\ \xi(t) < \eta(t) & \sum_{t'=t+t_{si/2}}^{\infty} (1 - \frac{\xi(t')}{\eta(t')}) \cdot t_{si/2} \end{cases}$$

The expected number of pilots ($\xi(t)$) can be estimated by any model for Provi-

sioning described through the related work [136, 12, 117]. They offer approximate results to the real ones achieved by the GWpilot PiF. Formulating a specific model fitted to PiF is extensive and it is postponed for a future work. In any case, $\xi(t')$ will be generally based on the maximum number of pilots (n) and on a cumulative distribution function (CDF), being $\xi(t') = n \cdot CDF(t')$, which will take values into $(0, n]$ because $t' = t_0 + t_{si/2} > 0$. On the other hand, for single applications $\eta(t)$ tends to 0 over time due to the end of calculation approaches. Therefore Υ will be content. Finally, the task turnaround without failures is:

$$\tau(t_0) = t_{si/2} + \Upsilon[t_0, t_{si/2}] + t_{pi} + t_x + t_e \tag{11.2}$$

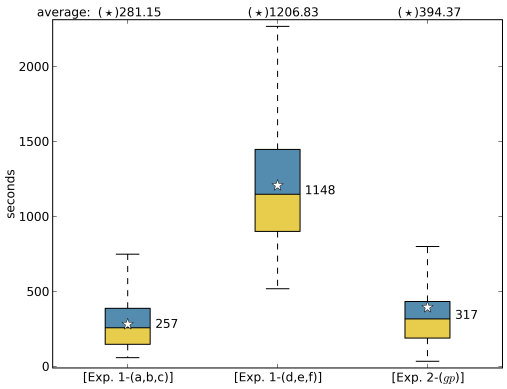
11.2.2. Statistical validation of the model

The accurateness of the fixed overheads ($t_{si/2}$ and t_{pi}) as well as the correctness of estimations (Υ) have to be demonstrated to validate the model formulated in this section. Experiments performed in this thesis, mainly in Chapter 9, indicate their suitability, but a mathematical proof was not shown. Therefore, an analysis using different approaches that comprises performance, scalability and overheads is made in this subsection. It is clear that these items can be only demonstrated increasing experimentally the stress of the system. Thus, statistics from the data obtained through the tests done in Chapters 7 and 10 were used because these experiments performed larger overloads. Collaterally, the study is useful to validate the design affirmations commented in Section 7.2 in comparison to other frameworks.

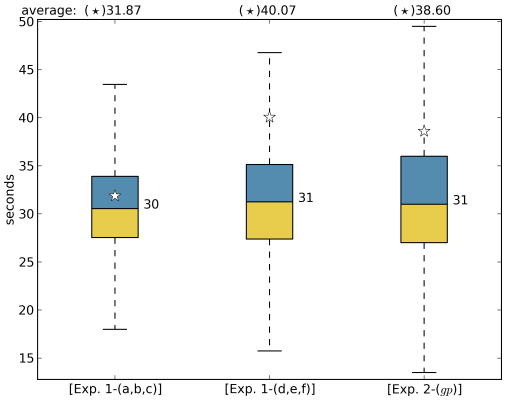
Performance, scalability and reliability

Firstly, the total number of processed user tasks in the experiments is an achievement itself if we have in mind that other systems consider these volumes [202, 196, 262] as the optimum throughput for months. In relation to Provisioning, the number of effectively enrolled pilots presented in Tables 7.4 and 10.3 is also important. There were tests with more than 9,000 successfully enrolled pilots [201], achieving maxima of above 200 pilots per minute [195]. However, it is interesting to provide the average number of enrolled pilots and the percentage of discarded pilots per hour [262], which are between 200-300 and above 5 %, respectively. Note that a discarded pilot does not imply that its hosting grid job has failed and vice versa. In any case, these values also show how the enrolment and discarding processes in GWpilot are faster and more effective than the mechanisms to provision resources used in other systems [202] which are based on middleware CLI.

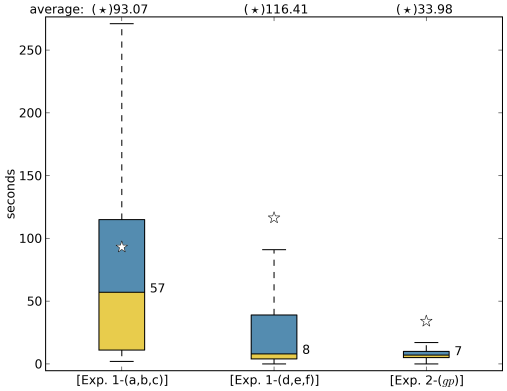
The dispatch capacity is another productivity and scalability aspect that must be examined. In the experiments, the time spent by the Scheduler was maintained below one second. Then GWpilot can effectively dispatch more than 500 tasks per minute. Other systems usually provide equivalent or lower rates. In any case, it is noteworthy that this dispatching aspect is indicative of the future scalability of GWpilot and its suitability for a wide range of calculations. In addition, Table 10.3 compiles the processing rate of pilot operations, whose values are easily manageable by any HTTP server (PiS is based on a lightweight HTTP framework). Therefore, the number of pilots can be potentially increased to greater orders of magnitude.



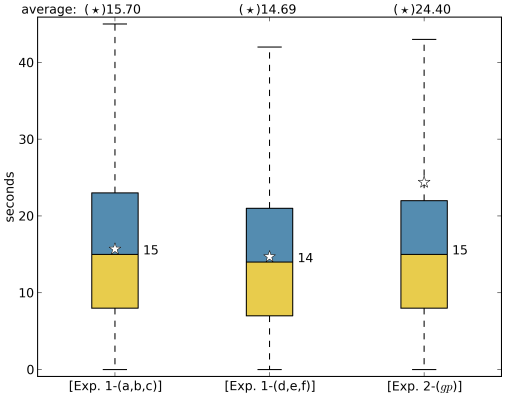
(a) Real execution time of tasks at remote resources (t_e).



(b) Pilot overhead (t_{pi}) and transfer times (t_x).



(c) Dispatching overhead, i.e. time spent in dispatching a task whenever it is included in the system ($t_{si/2} + \Upsilon[t_0, t_{si/2}]$, it depends on the number of pilots up).



(d) Time spent in generating and monitoring tasks for the applications (it is included even when it is not properly part of the task turnaround).

Figure 11.1: Box plots of the different turnaround times obtained with every completed task grouped by experiment. Every box stand for: [Exp. 1-(a, b, c)], i.e. the 3 short tests (a, b, c) of long multiplication corresponding to experiments performed in Chapter 7; [Exp. 1-(d, e, f)], i.e. the 3 long tests (d, e, f) of long multiplication corresponding to experiments also performed in Chapter 7; [Exp. 2-(*gp*)], i.e. the 3 tests with guided Provisioning (*gp*) performed in Chapter 10. Stars represent the average.

Turnaround and controlled overhead

Figures 11.1-(a, b, c) show three turnaround time values in the experiments presented in Chapters 7 and 10. They describe the turnaround of every task *done*, thus failure rates are not compiled. Tasks are grouped by experiment and subsequently decomposed in the three categories: CPU time (t_e), dispatching ($t_{si/2} + \Upsilon[t_0, t_{si/2}]$), and pilot and transfer overhead ($t_{pi} + t_x$).

Pilot operations have been extracted from T_{sched} to properly evaluate the overheads related to internal operations necessary to accept, classify, dispatch and consider the task as ended. Time wasted by internal operations is negligible compared with the time spent by the Scheduler, which is maintained below one second when there are

enough pilots. This fact is supported by the measurements previously made. Nevertheless, the turnaround increases when the Scheduler is unable to dispatch tasks due to the inexistence of idle pilots. Additionally, other experiments composed of higher volumes of tasks with complex requirements could increase this time. Therefore, Fig. 11.1-(c) allows demonstrating the accurateness of $t_{si/2}$ and the influence of Υ.

Usually, staging files are considered as part of the internal operations of the system [117] or even a separate overhead in other works [116]. In this study transfer times (t_x) are shown jointly to the pilot overhead (t_{pi}) to group the overheads related to the network in Fig. 11.1-(b).

Although it is neither concerned with the turnaround measurement nor with the GWpilot performance, the application overhead (T_{app}) is also displayed in Fig 11.1-(d) because it influences the final makespan of the calculation if there are not enough tasks provided to fill all the available pilots. This circumstance was described in Subsection 7.4.3, where a brief explanation was provided. Now, box plots indicate that the script that wraps the long multiplication application gives the similar performance when it supplies very short ($< lower\ limit >= 3$) or short ($< lower\ limit >= 9$) tasks. However, this overhead, along with the one originated in pilots (Fig. 11.1-(b)) and the configured *SCHEDULING_INTERVAL*, results in an important percentage with respect to CPU time in the first experiments. Therefore, *running* pilots are not completely filled.

CPU time (t_e) plotted in Fig. 11.1-(a) shows similar values for short experiments and the ones that together perform DKEsG and long multiplication calculations. The explanation is that the volume of multiplication tasks only represents approximately 11 % of all tasks in these experiments. Additionally, DKEsG-Mono tasks are shorter than very short multiplication tasks ($< lower\ limit >= 3$). Thus, the question is translated to the dispatching overhead shown in Fig. 11.1-(c). The maximum number of usable pilots is limited to the same maximum quantity of tasks in every experiment. An increase in the scheduling overhead indicates that either the number of *running* pilots is lower than 1,000, or some of these pilots are banned. Both circumstances explain the high values compiled for the short experiments and how they decrease when the duration of experiments is larger up to a median of 7 s (note that the *SCHEDULING_INTERVAL* is set to 10 seconds in every experiment). Thus, the dispatching overhead is actually describing the cost of Provisioning resources, but remains statistically stable if the number of tasks is adjusted to the number of *running* pilots. This is precisely one of the main performance measurements that should be analysed in this work: the variability of this overhead has not necessarily been taken into account by a self-scheduler that will use the GWpilot framework. $t_{si/2}$ can be enough to enable many algorithms to profit from the system.

The other important aspect is the pilot and transfer overhead ($t_{pi} + t_x$), whose medians are always close to 30 seconds because it mainly depends on its configured pulling interval (PI). Nevertheless, transferring files have influence on this overhead because the average values are higher in the experiments that deal with greater output sizes. This fact demonstrates the scalability of the GWpilot system, which is not influenced by the number of pilots running.

It is noteworthy to mention how the arithmetic average values of the corresponding measured parameters behave. In the figure describing the CPU time (Fig. 11.1-(a)), it can be seen almost a symmetric box with a higher weight in the distribution of the higher values, what can be appreciated in the average values (stars) and in the corresponding longer whiskers. Unlike the dispatching and pilot overheads, where the

averages lie outside the third quartile in the long first (< *lower limit* >= 9) and *gp* tests and even are outliers in the dispatching overhead, i.e. there have been some specific values of these overheads which deeply exceed the median and produce a non-symmetric average. This fact is not of importance if the median is symmetric inside the box, since it means that the distribution is almost homogenous and only some few tasks have produced a major overhead (pilot overhead case). However, it is noticeable an asymmetric distribution in the long first (< *lower limit* >= 9) and *gp* boxes of the scheduling overhead. In these cases, not only the average is an outlier, but also the lower values of the overhead are much concentrated around a value, i.e. most of the tasks (close to 50 %) have a low dispatching overhead.

11.3. Methodology to incorporate third-party schedulers

With the model presented and the GWpilot features, it is possible to easily separate the characterisation, the Application level scheduling, the Task Scheduling and the Provisioning, while maintaining the control over the pilot submission and removing the penalties originated by self-made estimations. However, it could be of interest for the developer to keep some of his implemented procedures to better fit the needs of his application. On the contrary, some users may not like to modify legacy codes to take advantage of pilots. Thus, a developer that plans to adapt any third-party scheduler to GWpilot should follow the following steps:

1. To identify the algorithm requirements (pre-conditions) that are related to characterisation, i.e. what qualifiers are necessary and what type of resources should be prioritised.

2. To check if GWpilot already offers procedures to accomplish these requirements and to value if they should substitute the possible legacy ones by:

   - Making a simple procedure to submit characterisation or customisation tasks to pilots whenever they were newly enrolled to the system.
   - Configuring the GWpilot Factory or to modify the legacy mechanism to submit pilot jobs.
   - Removing the compilation of statistical data from tasks (or even pilots) if accounting performed by the system is enough.

3. To purge the Workload Scheduling from useless procedures those have been achieved by the previous techniques and reformulate the proposed algorithm if needed.

4. To adjust the algorithm to establish equilibrium between spent time and profitable execution time taking into account the turnaround model and accordingly setting the GWpilot configuration.

Subsequent approaches

According to the decisions taken following the methodology, different possibilities appear. However, they can be grouped into the three main approaches set out below.

Each option progressively requires more work on adaptation of the personal scheduler code:

a) Straightforward method: Provisioning is automatically handled by the GWpilot Factory while the third-party tool is running as any other legacy application on the system, although the tool only submits tasks to pilot resources. In general, as pilots are stored as any other provider in the Host Pool, this methodology only requires some few modifications in the legacy code, similar to the ones explained in Chapter 8. The third-party tool will use its characterisation techniques in every pilot as if it was any provider. However, if some early-binding techniques are not invalidated, they will be applied on pilots and can impact the performance, resulting in overheads (for example, tasks with long suspension timeouts for inexistent batch queues).

b) Personal Provisioning: pilot jobs are submitted by the third-party tool making use of its own early-binding techniques. In this case, the tool must distinguish between providers and pilots by checking the *LRMS_NAME* tag. Moreover, as the selection of better providers is usually associated with a customised characterisation of the resource, the tool must take into account the pilot-provider relationship. For this purpose, pilots also publish their provider identifier into the *SITE_NAME* tag.

c) Totally-guided mechanism: GWpilot features are fully configured to satisfy all Provisioning matters with a similar or improved way from the ones offered by the third-party tool. This implies to purge the scheduler code from any early-binding or characterisation technique that can be replaced by a mechanism already offered by GWpilot. Therefore this option is the most expensive in terms of coding effort if a legacy scheduler is adapted, but it is specially recommended for incorporating new algorithms to the pilot system from scratch.

In general, these three approaches are combined to accomplish concrete needs.

11.4. Stacking self-schedulers on the cloud

To demonstrate how the proposed model and methodologies are suitable for incorporating external scheduling algorithms, even for those already included in legacy software, a good example should be the adaptation of a self-scheduler to GWpilot. For this purpose, the self-scheduling framework developed in Chapter 6 has been used. Therefore, the DyTSS algorithm will be utilised as a proof of concept to show how a loop-scheduler can be adapted to GWpilot.

It could seem that as the framework relies on GridWay, the procedure presented is only bounded to frameworks that use this platform [20, 132]. However, the *computation* is mainly based on DRMAA and other applications implemented with standards should be straightforwardly adapted too. Moreover, other frameworks that use different GSs or batch managers can be also easily adapted by substituting some commands, because getting information about resources and managing jobs is very simple with GWpilot.

Furthermore, the totally-guided approach was selected because it is the theoretically most expensive, but it allows profiting from all the GWpilot features.

11.4.1. Adaptation approach

DyTSS algorithm requires the profiles of the MC application and the updated benchmarks of every resource as well as its real availability (number of slots and reliability).

The characterisation mechanisms implemented in the self-scheduler are oriented to submit jobs directly to grid sites. For this purpose, replication was used for testing the slot availability, and an exhaustive accounting is performed by inspecting job outputs. However, these techniques are unnecessary in the network overlay created over the cloud resources. GWpilot features can be fully configured to satisfy all provisioning matters without replication, as well as characterisation tasks can be submitted to pilots and the accounting of GridWay can be finally used, avoiding the necessity of compiling statistics. In this sense, the benefits of continuously testing the providers can now be achieved by the correct configuration of GWpilot Factory and the banning feature, as has been explained in previous chapters.

Pilots provisioned are seen by the self-scheduler as suitable resources. However, these pilots have to be characterised before being profited. On the other hand, the application has to be profiled according to the benchmarks used for the latter to enable the matchmaking among them. For this purpose, the self-scheduler creates a set of profiling tasks that only require be executed in pilots. Consequently, GWpilot Factory detects these tasks and submits the necessary pilots. When tasks end, C_{eff} and S_{eff} are obtained.

To benchmark pilots and to force the Provisioning of certain volume of them, characterisation tasks are submitted. For this reason, self-scheduler maintains a set of *active* or *pending* characterisation tasks in the GWpilot framework to obtain the maximum number of pilots as possible, which is limited by the configuration of the Factory. Unlike the profiling tasks, the interest is now focused on the *performance* (P) and the *bandwidth* (BW) of every pilot. To obtain the information, the benchmark tool is executed again, but the big file does not copied, because pilots already publish information about bandwidth and latency. The analysis is always performed when a new pilot is detected and it publishes the results as an additional tag in the pilot. To preserve the accurateness of the monitoring, the values published are updated as any conventional task that belongs to the real calculation ends, as will be explained below. Moreover, characterisation tasks only will be executed in pilots that were not previously benchmarked, while conventional tasks only will run in pilots that publish the corresponding tag. This was performed setting the corresponding constraint in the task description.

It is noteworthy to mention that not only the allowed number of VMs in every cloud provider is variable, even the hardware provided by each resource is also so because it can be composed by different kinds of nodes and the provider can be overbooked. Therefore, the basis of the characterisation should rely on the same mathematical basis used for grid, but adapted to the turnaround model proposed for pilot jobs.

However, to adapt DyTSS it is not needed to take into account the variability in dispatching (i.e. $\Upsilon[t_0, t_{si/2}] \simeq 0$). Therefore, combining the Equations 6.1 and 11.1, the aforementioned parameters are used to estimate the turnaround time ($\tau(p_j, s)$) that substitutes the one ($T(r_j, s)$) stated in Section 6.2.1:

$$T(r_j, s) \simeq \tau(p_j, s) = t_{si/2} + t_{pi} + \frac{D}{BW_{p_j}} + \frac{C_{eff} + s \cdot S_{eff}}{P_{p_j}} \tag{11.3}$$

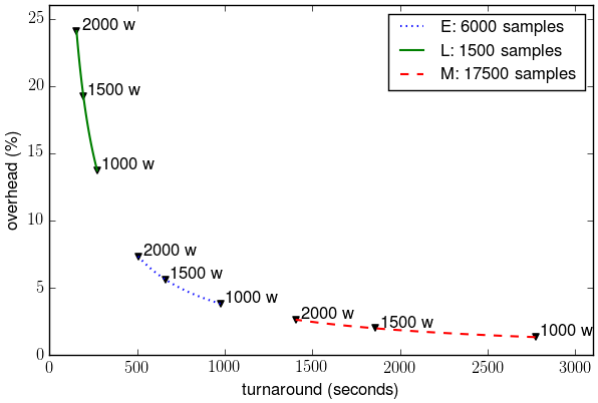


Figure 11.2: Overhead with respect to expected turnaround (Equation 11.3) according to the number of Nagano's samples per task and the GWpilot configuration ($t_{si/2} = 5$ s, $t_{pi} = 30$ s). An idealised infrastructure with $D/BW_{p_j} = 2$ s and without failures is considered, but cloud providers offer resources with benchmarked power (P) from 1,000 to 2,000 *whetstones* (w).

However, this model is unreliable if benchmarking parameters are not updated through the time. This is the reason for which every successful execution of a MC task is analysed to re-calculate them, so knowledge about the infrastructure is enhanced in real time with minimum computational effort. Moreover, DyTSS can receive now the exact benchmark of every slot effectively appropriated with the GWpilot system. This increases the efficiency of the algorithm. Additionally, the overheads introduced are reduced because DyTSS has to wait neither for benchmarks, nor for storing statistical data.

Therefore, as the accurateness of the model is preserved through the changes in the availability of resources, it can be used by the adaptive algorithm to determine the global performance of the virtual infrastructure provisioned and to consequently adapt the number of samples (s) to submit and where to do it.

11.4.2. Proposed tests

To evaluate the effectiveness of the model to incorporate advanced scheduling algorithms into the late-binding approach offered by GWpilot and to show how this incorporation is beneficial to calculations, even on cloud, the execution of Nagano [72] is selected to be carried out by the stacked framework in FedCloud.

As is explained in the Section B.3.1 of the Appendix B, the code is devoted to simulate fluorescence emissions and the energy deposited by electrons inside an observation volume of the desired proportions. The calculation for a single electron takes only a fraction of second, but for a real world use case, a complete simulation comprises several millions of electrons, for example $2 \cdot 10^7$. In this sense, its execution can be characterised following the profiling obtained in experiments in Subsection 6.3.2, resulting in C_{eff} =375.07 w and S_{eff} =156.26 w, being w *whetstone* units. The inputs and the application itself take 500 KB, while output requires only few KB, then the stage-in and -out process is really ballasted by the negotiation of the connection, lasting less than 2 seconds in current research or business networks.

Figure 11.2 shows the estimation of the overhead evolution for a task when it

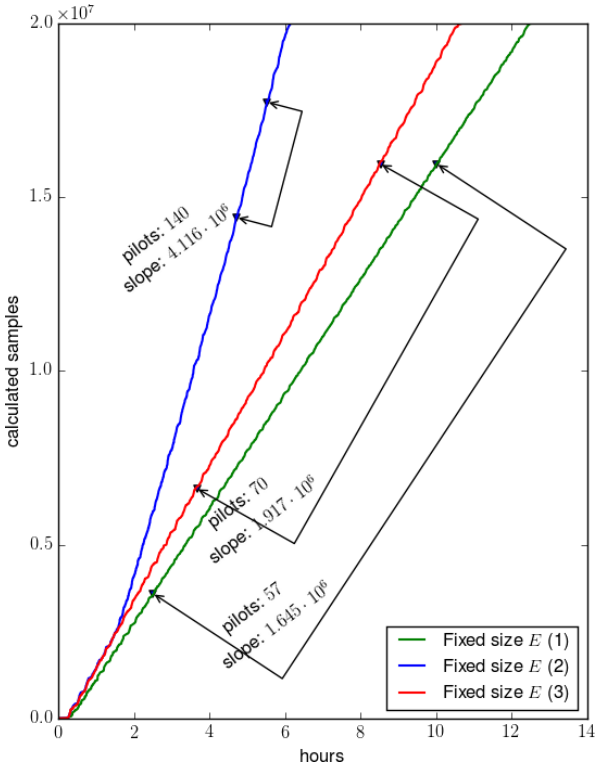


Figure 11.3: Nagano production using tasks with fixed size E of 6,000 particles (without DyTSS).

is executed on certain provisioned pilot, using the turnaround model proposed in this work and taking into account that resources offered by current cloud providers usually ranges from 1,000 w to 2,000 w. Each number of samples used coincides with the ones selected for the proposed tests. These are the limits L and M for DyTSS and the fixed number of samples E for an equal-sample-size distribution. As can be seen, these values are not arbitrary chosen. The overheads of the calculation based on equal-sized tasks are theoretically bounded into a 4-7 %, according to the turnaround model. In addition, the calculation should be few influenced by the failures of tasks due to their short duration (between 10 and 20 minutes). On the other hand, DyTSS has to reduce the makespan by dealing with very short and long tasks, i.e. achieving equilibrium among overheads, failures and turnaround to increase the production of samples.

Currently, the real availability of resources is very limited in FedCloud. Although a virtual image widely deployed in providers were used such as in the experiments in Section 8.4, the maximum number of VMs provisioned varying from few tens to hundred fifty, distributed among about seven reliable providers. This issue constraints the tests performed and the comparison to the results obtained in Chapter 6 following the early-binding approach. Consequently, two types of tests have been performed. The first one is the calculation of $2 \cdot 10^7$ samples divided in 3,333 tasks with fixed E size. The experiment is repeated three times to study the sample production according to the number of pilots provisioned without staking the self-scheduler. For the last one,

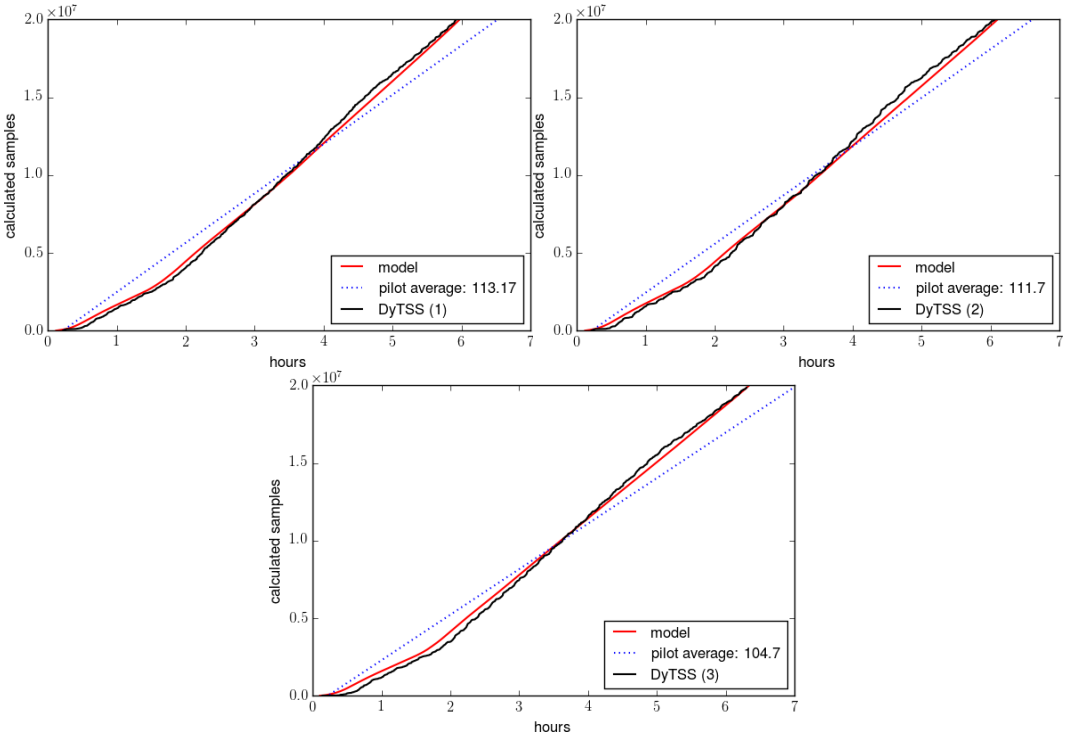


Figure 11.4: Nagano production with DyTSS algorithm.

the self-scheduler was stacked to GWpilot/GWcloud to perform the same calculation another three times, but using the aforementioned limits L and M. The idea is use the conclusions made in first experiment to analyse the reliability added by DyTSS to the calculation, independently of the number of pilots. These experiments have been launched at the same hour in different days.

11.4.3. Results

Regarding how the system behaves when the self-scheduler is not stacked, it can be seen in Figure 11.3 that the number of calculated samples increases as the number of pilots do, as expected. Slopes are shown during the longer interval in which the number of pilots is maintained constant in the experiment. In this sense, a conclusion can be found if how the slope (i.e. number of calculated samples) increases with the number of pilots is analysed. Fitting to a new linear curve the number of pilots and the slope, it is found that the number of calculated samples increases in a factor of ~ 32 as the number of pilots do (this fitting presents a correlation factor r of 0.999 and a R^2 coefficient of 0.998). On the other hand, the linear regression between the interval points where the pilots remain stable coincides with the performance of the provisioned infrastructure using this sample distribution [110]. The heuristic of DyTSS is based on calculating this performance for different distributions, and therefore it can be used to verify if the self-scheduler achieves better results than a fine equal-sized choice. It is noteworthy to mention that the type provisioned resources vary among experiments, but the slopes are similar when the number of pilots is maintained stable.

As can be seen in Figure 11.4, when the self-scheduling framework is stacked, the coupled tool outperforms the simple GWpilot/GWcloud system in the long term, although the latter had provisioned whole pilots previously the experiment starts. To found this conclusion, the productivity of the experiment with fixed size E has been estimated by calculating the slope for a number of pilots equivalent to the real average of pilots provisioned when the self-scheduler is used. In the beginning, the former experiment executes better as its lineal behaviour starts calculating samples more quickly. During those initial hours, the stacked system presents an exponential behaviour as the proper provision takes longer, but from one point on (around 3 hours and 45 minutes), it is able to calculate more samples. The result is that experiments without the self-scheduler would last $\sim$ 8-13 % more.

Figure 11.4 also shows the behaviour of the DyTSS algorithm. First tasks are bigger than last ones. This is done to reduce the overheads during the most part of the calculation and limits the influence of failed tasks at the end. However, this fact decreases the productivity at the beginning, because many tasks continue running without returning results. The only way to describe this issue is drawing the sample production as the system were working with a task size between L and M, for example E. As the number of pilots and their benchmark average have been monitored every minute, the turnaround model can be used for this purpose. Thus, in Figure 11.4 it is also depicted the proposed mathematical model (red line). Using this reference it can be seen that DyTSS works first under that line and then above it. Furthermore, it clearly demonstrates that the proposed model fits the real execution of the pilots in a real cloud infrastructure as FedCloud is. This fact is key to foresee the behaviour of a cloud infrastructure for scheduling and provisioning resources in advance.

11.5. Conclusions

The last requirement stated in Subsection 3.2.2 for the fully profiting from the Multilevel Scheduling has been achieved in this chapter. That is, a simplified turnaround model, which can be used by third-party schedulers, has been formulated based on a statistical study of the overheads generated by GWpilot carrying out real calculations on production infrastructures. Moreover, a methodology to incorporate these third-party tools into GWpilot has been presented. Therefore, unlike other systems, the new pilot framework is capable of including diverse scheduling algorithms such as the ones compiled in Subsection 2.3.3. Doing so, the personalised characterisation that those scheduling algorithms require becomes feasible, a fact that overcomes their lack of trustworthiness in the information provided by the grid and cloud services.

The suitability of the approach has been demonstrated by stacking the legacy self-scheduler developed in Chapter 6 to the pilot framework. For this purpose, the execution of Nagano application relying on this coupled system has been compared with the standalone use of GWpilot on the EGI FedCloud infrastructure. The accurate characterisation of overheads and resources allows self-scheduler to properly perform a reliable MC execution based on its Workload Scheduling algorithm, which results in an improved makespan. Furthermore, the proposed model perfectly matches the real production of the legacy self-scheduler stacked to the pilot system. The overhead produced with respect to the expected turnaround has been analysed as well.

Chapter 12

Main Contributions and Future Work

12.1. Contributions and expected impact

The main intellectual contribution of this research is the bestowed capacity to fully exploit the Multilevel Scheduling advantages to users, developers and institutions. This achievement allows them to actually overcome their real computational challenges on distributed environments. This fact has been extensively demonstrated performing numerous meaningful calculations for different knowledge areas and computational needs, which have required making the most of a huge volume of resources belonging to real grid and cloud infrastructures in production.

The main expression of the previous contribution is the proposed design of a new framework, GWpilot, which fully profits from the advantages of pilot jobs to build a usable and adaptable Multilevel Scheduling architecture, suitable for improving the execution of a wider range of distributed applications on grids and clouds. This constitutes a remarkable step forward in the profiting of large distributed computing infrastructures, which had not really been achieved before by any other pilot system due to lack of adaptability, compatibility and deploy-ability. In addition, the extensive study performed to design GWpilot has resulted in a complete taxonomy of pilot systems that can be used as a guideline for future researches, developments and deployments.

The success of the new approach is based on the solutions provided by the diverse new technologies and methodologies that have been developed through this thesis. First, the system properly tackles the persistent problem of characterisation that traditionally has prevented the deployment of complex scheduling algorithms or policies on grid and cloud environments. It provides users with the needed tools to arbitrarily incorporate a customised monitoring of resources and their running applications into the system. Thus, the user can easily take advantage of this feature to perform a specialised scheduling of his application workload without the need of modifying any code in the pilot system. Moreover, few changes in their applications are needed to manage the monitoring and scheduling, because the framework supports legacy applications, basic scheduling policies based on the description of every task, and fairness in a shared and competitive environment.

Users can also automatically manage the Resource Provisioning among different grid and cloud providers without the need of explicitly indicating one resource or manually submitting pilots. Moreover, the post-configuration of the provisioned workspaces is allowed, especially in cloud, without the need of dealing with contextualisation. For this purpose, the compatibility and security with different infrastructures have been preserved and extended. In this sense, a new brokering approach was proposed for the dynamic Provisioning of virtual workspaces in clouds that is suitable for distributed calculations and allows users to select the virtual environment that their application requires. These distinguishing features are not offered by other pilot systems. Moreover, the lack of feasible cloud brokering mechanisms for High Throughput Computing has been overcome with this approach, constituting another important contribution.

Furthermore, the capacity to customise whole scheduling layers in the proposed Multilevel architecture is raised to the user-level. Additionally, a new and simplified mathematical model has been formulated to accurately estimate the turnaround of every task. These achievements benefit skilled developers and administrators, who can now build complex scheduling policies, a capacity that was not actually provided by either other pilot systems or with early-binding mechanisms. Even more, the combination of advances presented in this thesis allows the easy adaptation of legacy third-party scheduling tools to the pilot framework such as self-schedulers and workflow managers. This goal was not registered before and it enables profiting from the compiled knowledge and previous work done on grid environments for fifteen years.

To demonstrate these achievements, the new pilot framework has been implemented and tested with different legacy applications and scheduling policies and systems, performing meaningful calculations on cloud and grid infrastructures in production. For this purpose, customary applications belonging to different knowledge areas have been adapted to run on distributed environments. Additionally, the support of cloud brokering is achieved. Moreover, a self-scheduling framework has been developed to demonstrate the advantages of stacking third-party schedulers onto the pilot system. The new proposed technologies have been previously tested following the early-binding approach to enable the subsequent comparison with the pilot framework. Therefore, these developments implied an extensive study of early-binding techniques and their limitations that has been compiled in Part I of this thesis, and which constitutes another intellectual contribution.

Moreover through the Part II of this thesis, it has been experimentally demonstrated that the new framework outperforms the performance of other pilot systems due to its design, even if its advanced scheduling capacities were disabled. Such a result has been demonstrated on real infrastructures and performing real calculations. In addition, the fully support of legacy applications has been illustrated as well as the transparent profiting of brokering in cloud federations. Experiments have been performed to show how any user can straightforwardly make the most of the power of cloud with GWpilot. The framework also maintains its performance and scalability when multiple users and applications are being scheduled, independently of their volume or the short duration of the tasks, even when complex scheduling policies are performed. In this sense, the methodologies to customise the Multilevel Scheduling architecture have been extensively described and tested with competitive applications scheduled at the same time. The validity of the proposed mathematical model has been statistically demonstrated via measurements of real executions; its suitability for being used as the basis of the adaptation of third-party schedulers has been demonstrated as well.

For the latter, the developed self-scheduler has been stacked on GWpilot, obtaining an improved performance.

The research performed in this thesis is focused on solving the real problems of more quickly obtaining much accurate scientific results by means of grid and cloud approaches. Thus, the experiments carried out have also provided valuable biological and physical results. The main ones are the study of the relation between rotational transform scaling and Neoclassical transport in stellarators (Section C.4.3) and the comparison of Neoclassical fluxes (Section C.4.2) in the TJ-II device. Additionally, other contributions to other fields have been done (Appendix B). For example, the comparison among genotypes of several human viruses has allowed to know their relationship and mutations. New studies on superconducting vortex dynamics have been also achieved with the experiments performed in this thesis.

The new pilot framework is available to be profited by the scientific and industrial communities. As it has been demonstrated through this thesis, the framework can be easily deployed and improved. Therefore, the impact of this thesis on Computer Science as well as on any other field is guaranteed by the practicality, extensibility, and the multiple possibilities of scheduling that the proposed approach allows.

12.2. Future work

The demonstrated usability, adaptability, and compatibility of the proposed framework and, overall, its capacity to really support complex scheduling in real distributed environments, opens the door to novel approaches that were difficult or even impossible to be carried out with previous technologies.

First, Cloud Computing and Big Data are experiencing a fast development and expansion. Through the next years, federations and public computational markets based on both paradigms will become the most powerful platforms available at the expense of grid infrastructures. Consequently, many customary distributed applications have to be adapted to these environments and GWpilot is the way of achieving this challenge. For this purpose, some applications adapted as part of this thesis, such as FLUKA and GAMOS, are currently being exploited on cloud [242], but others may be adapted too. In this sense, the simulation of fusion reactors continues to be an important scientific challenge and several studies are being carried on cloud with other kind of codes.

However, additional scopes are being considered. The recently incorporation of CIEMAT as full member of the Latin American Giant Observatory (LAGO) Collaboration allows opening new lines of research. In LAGO, it is necessary to access great amount of data, which have to be processed taking into account their much different geographical locality. Thus, technologies close to Big Data have to be developed. It will be easier to deploy the required Big Data services and algorithms into clouds with GWpilot.

It is noteworthy to mention again that a great percentage of future providers will follow a pay-per use policy, despite of the scientific federations established. Therefore, the incorporation of cloud resources into the framework is being evaluated considering the economical questions that arise in an environment with multiple commercial providers. Moreover, the scheduling based on QoS, budgets and deadlines, taking into account the consumed CPU, I/O and network operations or wasted storage as well as the performance obtained will be a must.

Other research area with projection is the Autonomous Computing field. In this thesis, some advances have been presented to enable resilient scheduling algorithms in clouds. However, to make progress on the self-optimisation of systems implies the creation of novel scheduling algorithms based on advanced heuristics, Soft Computing and other approaches such as machine learning, which allow foreseeing the future failures and consequently adapting the execution in advance. Moreover, these scheduling can be improved with the management of checkpoints. The capacity to suspend, migrate and restart a calculation composed by a huge volume of processes before failures happen or due to priority reasons is one of the main objectives of the Exascale challenge in the next decade. For this reason, the GWpilot support of checkpointing is being tested to assure the compatibility with the future Exascale platforms.

Furthermore, there is room to extend the framework with many other functionalities and purposes. For example the usability can be improved by implementing a graphical interface, by coupling a science gateway or by using a workflow manager that should be remotely accessed. In this sense, several approaches for remotely accessing GWpilot through OGSA-BES standard have been tested with workflow managers [263, 264] and only the consolidation of the work performed is still needed.

The framework it is also being currently extended to fully enable pilots with the multi-task capacity. This feature will allow profiting from several processors in the same provisioned virtual machine or even in commodity hardware to build a cluster without the need of managing a complex LRMS. Moreover, the access to underlying hardware of accelerators such as GPUs or Xeon Phis, as well as their future support by virtualisation hypervisors, will increase the computational power and versatility of the proposed solution. Thus, the future work will be mainly focused on mixed distributed and parallel calculations. In this sense, workflows of fusion applications that require strong communication among their tasks, like those used in turbulence simulations, are being explored. They include calculations based upon Gyrofluid or Gyrokinetic Theory, which are challenging in the sense of being able to simulate real transport time scales. In any case, the incorporation of any other complex scheduling algorithm to accomplish the requirements of specific workload problems will be explored on-demand to achieve the computational challenges of the scientific projects involved.

Therefore, the future suitability and sustainability of the approach presented in this thesis are granted as well as the continuity of the research.

Part III

APPENDICES

Appendix A

Dissemination

The whole scientific contribution of this thesis can be summarised in a set of publications and dissemination activities that have to be classified in order to provide a clear view of the quality of the performed research. This classification is shown in this appendix with the customary impact and quality measurements used by the scientific community.

A.1. JCR publications

The research performed in this thesis has resulted in eight articles published in journals indexed in Journal Citation Reports (JCR): two in quartile 1 (Q1), one in quartile 2 (Q2), three in quartile 3 (Q3) and two in quartile 4 (Q4). Three out of total (Q1, Q3 and Q4) are in different reviewing phases.

1. A. J. Rubio-Montero, E. Huedo, and R. Mayo-García, "Scheduling multiple virtual environments in cloud federations for distributed calculations," *Future Generation Computer Systems*, p. (Under Review), 2016.

    - JCR (2014) IF: 2.786. 5-year IF: 2.464. 8/102 (Q1) in COMPUTER SCIENCE, THEORY & METHODS.
    - Number of pages: 15, double column.
    - References: 52.

2. A. J. Rubio-Montero, M. A. Rodríguez-Pascual, and R. Mayo-García, "A simple model to exploit reliable algorithms in cloud federations," *Soft Computing*, p. (Under Review), 2016.

    - JCR (2014). IF: 1.271. 5-year IF: 1.635. 65/123 (Q3) in COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE and 58/102 (Q3) in COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS
    - Number of pages: 13, double column.
    - References: 50.

3. F. Castejón, A. J. Rubio-Montero, A. López-Fraguas, E. Ascasíbar, and R. Mayo-García, "Neoclassical transport and iota scaling in the TJ-II stellarator," *Fusion Science and Technology*, p. (Accepted), 2016.

- JCR (2014). IF: 0.486. 5-year IF: 0.480. 28/34 (Q4) in NUCLEAR SCIENCE & TECHNOLOGY.
- Number of pages: 35, single column.
- References: 34.

4. A. J. Rubio-Montero, F. Castejón, E. Huedo, and R. Mayo-García, "A novel pilot job approach for improving the execution of distributed codes: application to the study of ordering in collisional transport in fusion plasmas," *Concurrency and Computation: Practice & Experience*, vol. 27, no. 13, pp. 3220–3244, September 2015. doi : 10.1002/cpe.3301

   - JCR (2014) IF: 0.997. 5-year IF: 0.958. 47/104 (Q2) in COMPUTER SCIENCE, SOFTWARE ENGINEERING, and 46/102 (Q2) in COMPUTER SCIENCE, THEORY & METHODS.
   - Number of pages: 25, single column.
   - References: 44.
   - Cited by: 1 (Google Scholar)

5. A. J. Rubio-Montero, E. Huedo, F. Castejón, and R. Mayo-García, "GWpilot: Enabling multi-level scheduling in distributed infrastructures with GridWay and pilot jobs," *Future Generation Computer Systems*, vol. 45, pp. 25–52, April 2015. doi : 10.1016/j.future.2014.10.003

   - JCR (2014) IF: 2.786. 5-year IF: 2.464. 8/102 (Q1) in COMPUTER SCIENCE, THEORY & METHODS.
   - Number of pages: 28, double column.
   - References: 81.
   - Cited by: 5 (Google Scholar)

6. M. Rodríguez-Pascual, A. Gómez, R. Mayo-García, D. P. de Lara, E. M. González, A. J. Rubio-Montero, and J. L. Vicent, "Superconducting Vortex Lattice Configurations on Periodic Potentials: Simulation and Experiment," *Superconductivity and Novell Magnetism*, vol. 25, no. 7, pp. 2127–2130, October 2012. doi : 10.1007/s10948-012-1636-8

   - JCR (2014) IF: 0.909. 5-year IF: 0.759. 112/144 (Q4) in PHYSICS, APPLIED and 53/67 (Q4) in PHYSICS, CONDENSED MATTER.
   - Number of pages: 4, double column.
   - References: 14.
   - Cited by: 1 (Google Scholar).

7. A. J. Rubio-Montero, F. Castejón, M. A. Rodríguez-Pascual, E. Montes, and R. Mayo, "Drift Kinetic Equation Solver for Grid (DKEsG)," *IEEE Transactions on Plasma Science*, vol. 38, no. 9, pp. 2093–2101, September 2010. doi : 10.1109/TPS.2010.2055164

   - JCR (2014) IF: 1.101. 5-year IF: 1.089. 21/31 (Q3) in PHYSICS, FLUIDS & PLASMAS.

- Number of pages: 9, double column.
- References: 25.
- Cited by: 6 (Google Scholar).

8. M. Rodríguez-Pascual, J. Guasp, F. Castejón, A. J. Rubio-Montero, I. M. Llorente, and R. Mayo, "Improvements on the Fusion Code FAFNER2," *IEEE Transactions on Plasma Science*, vol. 38, no. 9, pp. 2102–2110, September 2010. doi : 10.1109/TPS.2010.2057450

   - JCR (2014) IF: 1.101. 5-year IF: 1.089. 21/31 (Q3) in PHYSICS, FLUIDS & PLASMAS.
   - Number of pages: 9, double column.
   - References: 24.
   - Cited by: 8 (Google Scholar).

A.2. Book chapters and other journals

This section compiles the peer-reviewed contributions that are not indexed in JCR.

1. M. Rodríguez-Pascual, C. Kanellopoulos, A. J. Rubio-Montero, D. Darriba, O. Prnjat, D. Posada, and R. Mayo-García, "Adapting reproducible research capabilities to resilient distributed calculations," *International Journal of Grid and High Performance Computing*, p. (Accepted), 2016.

   - Number of pages: 13, single column.
   - References: 23.

2. R. Isea, E. Montes, A. J. Rubio-Montero, and R. Mayo, *State-of-Art with PhyloGrid: Grid Computing Phylogenetic Studies on the EELA-2 Project Infrastructure*, in *Grid Computing: Towards a Global Interconnected Infrastructure*, ser. Computer Communications and Networks. Springer London / Heidelberg New York, 2011, pp. 277–291. doi : 10.1007/978-0-85729-676-4_11

   - Acceptation rate: 17.6 % (27/153).
   - Number of pages: 15, single column.
   - References: 30.

A.3. Proceedings

Manuscripts presented in conferences and published in proceeding books are listed in this section, ordered by year. The core CORE/ERA ranking[1] is mainly used by the scientific community and certification authorities to know the quality and possible impact of conferences on Computer Science. Every conference is ranked by: (A*) flagship conference, (A) excellent conference, (B) good conference and, (C) conference that meets minimum quality standards. In this sense, the research performed in this

[1] http://www.core.edu.au/index.php/conference-rankings

thesis has resulted in fourteen proceedings, of which one is classified as A, four as B and three as C. The quality of the remaining six is supported by the publisher and/or editor/committee or by other measurements as citations.

1. A. J. Rubio-Montero, E. Huedo, and R. Mayo-García, "User-Guided Provisioning in Federated Clouds for Distributed Calculations," in *Adaptive Resource Management and Scheduling for Cloud Computing (ARMS-CC 2015)*, ser. Lecture Notes in Computer Science, vol. 9438. San Sebastián, Spain: Springer, 20th July 2015, pp. 60–77. doi : 10.1007/978-3-319-28448-4_5

   - Number of pages: 18, single column.
   - References: 37.

2. A. J. Rubio-Montero, M. A. Rodríguez-Pascual, and R. Mayo-García, "Evaluation of an adaptive framework for resilient Monte Carlo executions," in *30th ACM/SIGAPP Symposium On Applied Computing (SAC'15)*. Salamanca, Spain: ACM New York, 13–17 April 2015, pp. 448–455. doi : 10.1145/2695664.2695890

   - ERA/CORE: B (Field Of Research: 0804 - Data Format).
   - Number of pages: 8, double column.
   - References: 29.

3. M. Rodríguez-Pascual, A. J. Rubio-Montero, R. Mayo-García, C. Kanellopoulos, O. Prnjat, D. Darriba, and D. Posada, "A fault tolerant workflow for reproducible research," in *Annual Global Online Conference on Information and Computer Technology (GOCICT 2014)*. Louisville, Kentucky, USA: IEEE CS Press, 3–5 December 2014, pp. 70–75. doi : 10.1109/GOCICT.2014.10

   - Number of pages: 6, double column.
   - References: 15.

4. A. J. Rubio-Montero, F. Castejón, E. Huedo, M. Rodríguez-Pascual, and R. Mayo-García, "Performance improvements for the neoclassical transport calculation on Grid by means of pilot jobs," in *Int. Conf. on High Performance Comput. and Simulation (HPCS 2012)*. Madrid, Spain: IEEE CS Press, 2–6 July 2012, pp. 609–615. doi : 10.1109/HPCSim.2012.6266981

   - ERA/CORE: B (Field Of Research: 0805 - Distributed Computing).
   - Number of pages: 7, double column.
   - References: 31.
   - Cited by: 1 (Google Scholar).

5. M. Rodríguez-Pascual, A. J. Rubio-Montero, R. Mayo, A. Bustos, F. Castejón, and I. Llorente, "More Efficient Executions of Monte Carlo Fusion Codes by Means of Montera: The ISDEP Use Case," in *19th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2011)*. Ayia Napa, Cyprus: IEEE CS Press, 9–11 February 2011, pp. 380–384.doi:10.1109/PDP.2011.46

   - ERA/CORE: C (Field Of Research: 0805 - Distributed Computing).

- Number of pages: 5, double column.
- References: 15.
- Cited by: 8 (Google Scholar).

6. A. J. Rubio-Montero, L. Flores, F. Castejón, E. Montes, M. Rodríguez-Pascual, and R. Mayo, "Executions of a Drift Kinetic Ecuation solver on Grid," in *18th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2010)*. Pisa, Italy: IEEE CS Press, 17–19 February 2010, pp. 454–459. doi : 10.1109/PDP.2010.40

   - ERA/CORE: C (Field Of Research: 0805 - Distributed Computing).
   - Number of pages: 6, double column.
   - References: 14.
   - Cited by: 2 (Google Scholar).

7. M. Rodíguez-Pascual, D. P. de Lara, E. M. González, A. Gómez, A. J. Rubio-Montero, R. Mayo, and J. Vicent, "Grid computing simulation of superconducting vortex lattice in superconducting magnetic nanostructures," in *Proceedings of the 4th Iberian Grid Infrastructure Conference*, vol. 4. Braga, Portugal: NETBIBLO S.L. (Sta. Cristina, La Coruña, Spain), 24–27 May 2010, pp. 97–109. ISBN 978-84-9745-549-7

   - Number of pages: 12, single column.
   - References: 14.

8. M. A. Rodríguez-Pascual, J. Guasp, F. Castejón, A. J. Rubio-Montero, I. M. Llorente, and R. Mayo, "A Grid version of the Fusion code FAFNER," in *18th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2010)*. Pisa, Italy: IEEE CS Press, 17–19 February 2010, pp. 449–453. doi : 10.1109/PDP.2010.37

   - ERA/CORE: C (Field Of Research: 0805 - Distributed Computing).
   - Number of pages: 5, double column.
   - References: 10.

9. R. Isea, E. Montes, A. J. Rubio-Montero, J. D. Rosales, M. A. Rodríguez-Pascual, and R. Mayo, "Characterization of antigenetic serotypes from the dengue virus in Venezuela by means of Grid Computing," in *Healthgrid Applications and core Technologies. Proceedings of HealthGrid 2010*, ser. Studies in Health Technology and Informatics, vol. 159. Paris, France: IOS Press, 28–30 June 2010, pp. 234–238. doi : 10.3233/978-1-60750-583-9-234

   - Number of pages: 5, single column.
   - References: 22.
   - Cited by: 2 (Google Scholar).

10. M. Rodríguez-Pascual, F. Castejón, A. J. Rubio-Montero, R. Mayo, and I. M. Llorente, "FAFNER2: A comparison between the Grid and the MPI versions of the code," in *Int. Conf. on High Performance Comput. and Simulation (HPCS 2010)*. Caen, France: IEEE CS Press, 28 June–2 July 2010, pp. 78–84. doi : 10.1109/HPCS.2010.5547146

    - ERA/CORE: B (Field Of Research: 0805 - Distributed Computing).
    - Number of pages: 7, double column.
    - References: 19.
    - Cited by: 2 (Google Scholar).

11. R. Isea, E. Montes, A. J. Rubio-Montero, and R. Mayo, "Computational Challenges on Grid Computing for Workflows Applied to Phylogeny," in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living. 10th Int. Work-Conference on Artificial Neural Networks (IWANN 2009)*, ser. Lecture Notes in Computer Science, vol. 5518. Salamanca, Spain: Springer-Verlag, 10–12 June 2009, pp. 1130–1138. doi : 10.1007/978-3-642-02481-8_171

    - ERA/CORE: B (Field Of Research: 0801 - Artificial Intelligence and Image Processing).
    - Number of pages: 9, single column.
    - References: 21.
    - Cited by: 8 (Google Scholar).

12. A. J. Rubio-Montero, P. Arce, J. I. Lagares, Y. P. Ivanov, D. A. Burbano, G. Díaz, and R. Mayo, "Performance Tests of GAMOS Software on EELA-2 Infrastructure," in *Proceedings of the Second EELA-2 Conference*. Choroní, Venezuela: Editorial CIEMAT (Madrid, Spain), 25–27 November 2009, pp. 379–385. ISBN 978-84-7834-627-1

    - Number of pages: 7, single column.
    - References: 16.
    - Cited by: 2 (Google Scholar).

13. R. Isea, J. Chaves, E. Montes, A. J. Rubio-Montero, and R. Mayo, "The evolution of HPV by means of a phylogenetic study," in *Healthgrid Research, Innovation and Business Case. Proceedings of HealthGrid 2009*, ser. Studies in Health Technology and Informatics, vol. 147. Berlin, Germany: IOS Press, 29 June–1 July 2009, pp. 245–250. doi : 10.3233/978-1-60750-027-8-245

    - Number of pages: 6, single column.
    - References: 13.
    - Cited by: 1 (Google Scholar).

14. A. J. Rubio-Montero, R. S. Montero, E. Huedo, and I. M. Llorente, "Management of Virtual Machines on Globus Grids using GridWay," in *21st IEEE Int. Parallel and Distributed Processing Symposium (IPDPS 2007)*. Long Beach, USA: IEEE CS Press, 27–30 March 2007, pp. 1–7.doi:10.1109/IPDPS.2007.370548

- ERA/CORE: A (Field Of Research: 0805 - Distributed Computing).
- Number of pages: 7, double column.
- References: 16.
- Cited by: 29 (Google Scholar).

A.4. Other contributions

This section includes a selection of the contributions to highly respected conferences in their discipline area, such as oral presentations and posters, which have not resulted in proceedings and are usually included in books of abstracts. In this sense, EGI and EGEE forums were the main venue organised in Europe for researchers working on Grid Computing, being usually co-located with the Open Grid Forum standardisation meetings. On the other hand, the International Conference on Numerical Simulation of Plasmas (ICNSP) is a bi-annual meeting that started in 1967 with the aim to highlight major advances in computational Plasma Physics and related areas. While the IEEE Nuclear Science Symposium & Medical Imaging Conference (NSS-MIC) is one of the world' s leading multidisciplinary conferences focused on advancements in the fields of Nuclear Science and Software Engineering, such as radiation detection, data acquisition, or medical imaging applications. On the other hand, the International Society for Computational Biology (ISCB) is a scholarly society with nearly 2,000 members from over 50 countries, which has emerged as leader in the field of Computational Biology. Therefore, the following contributions increased the visibility of the research performed in this thesis.

1. A. J. Rubio-Montero, M. Plociennik, I. Marín-Carrión, T. Zok, M. Rodríguez-Pascual, R. Mayo-García, M. Owsiak, E. Huedo, F. Castejón, and B. Palak, "Advantages of adopting late-binding techniques through standardised interfaces for workflow managers," Poster. BoA of EGI Technical Forum 2013, Madrid, Spain, 16–20 September 2013.

2. M. Plociennik, A. J. Rubio-Montero, T. Zok, I. Marín-Carrión, M. Rodríguez-Pascual, R. Mayo-García, F. Castejón, E. Huedo, M. Owsiak, and B. Palak, "OGSA-BES connector for Kepler to remotely use the GridWay meta-scheduler," Poster. BoA of EGI Community Forum 2013, Manchester, United Kingdom, 8–12 April 2013.

3. M. Rodríguez-Pascual, A. J. Rubio-Montero, and R. Mayo-García, "An unattended, fault-tolerant approach for the execution of distributed applications," Oral presentation. BoA of EGI Community Forum 2013, Manchester, United Kingdom, 8–12 April 2013.

4. A. J. Rubio-Montero, E. Huedo, F. Castejón, J. Velasco, and R. Mayo-García, "New computational methodology for the execution of massive distributed calculations: Its application to the neoclassical transport in nuclear fusion plasmas," Poster. IEEE Nuclear Science Symposium and Medical Image Conference (NSS-MIC 2012), Anaheim, USA, 27 October – 03 November 2012.

5. A. J. Rubio-Montero, E. Huedo, and R. Mayo-García, "GWpilot: a personal (or institutional) pilot system ," Oral presentation. 3rd EGI Technical Forum, Contributions Book, Prague, Czech Republic, 18–20 September 2012.

6. M. Rodríguez-Pascual, A. J. Rubio-Montero, and R. Mayo, "Adaptive Scheduling," Oral presentation. International Society for Computational Biology (ISCB), Latin America 2012 Conference on Bioinformatics, (ISCB-LA 2012), Santiago de Chile, Chile, 17–21 March 2012.

7. M. Rodríguez-Pascual, A. J. Rubio-Montero, and R. Mayo, "More efficient Monte Carlo Grid Executions with Montera Framework," Oral presentation. IEEE Nuclear Science Symposium and Medical Image Conference. (NSS-MIC 2011), Valencia, Spain, 23 –29 October 2011.

8. A. J. Rubio-Montero, M. Rodríguez-Pascual, F. Castejón, E. Montes, and R. Mayo, "The Drift Kinetic Ecuation solver for Grid (DKEsG)," Oral presentation. 21st International Conference on Numerical Simulation of Plasmas (ICNSP 2009), Lisbon, Portugal, 6–9 October 2009.

9. M. Rodríguez-Pascual, J. Guasp, F. Castejón, A. J. Rubio-Montero, I. M. Llorente, and R. Mayo, "NBI heating simulations of fusion plasmas on the Grid (FAFNER2)," Poster. 21st International Conference on Numerical Simulation of Plasmas (ICNSP 2009), Lisbon, Portugal, 6–9 October 2009.

10. A. J. Rubio-Montero, E. Montes, M. Rodríguez, F. Castejón, and R. Mayo, "A Grid fusion code for the Drift Kinetic Equation solver," Poster. BoA of 4th EGEE User Forum, Catania, Italy, 2–6 March 2009.

11. M. Rodríguez, J. Guasp, F. Castejón, I. Llorente, A. J. Rubio-Montero, and R. Mayo, "Improvements on the EGEE fusion code FAFNER-2," Oral presentation. BoA of 4th EGEE User Forum, Catania, Italy, 2–6 March 2009.

Appendix B

Applications

Science and engineering have tremendously evolved in the last decades thanks to the application of numerical methods to the problems these fields were facing. In this way, not only the achievement of new solutions has become feasible, the problems themselves have turned more ambitious as new more powerful computing infrastructures were available. In this appendix, some and much different examples are outlined in which the solutions proposed in this thesis have been cornerstone for obtaining new scientific results.

B.1. Chemical Physics

As it is well known, Chemical Physics covers a wide range of phenomena that are in between the molecular and atomic levels. In such a scenario in which Thermodynamics, Quantum Mechanics, Kinetics and Statistical Mechanics play different roles, only computing simulations can approach the huge amount of particles that are involved. In a world in which the Avogadro constant ($6.023{\cdot}10^{23} mol^{-1}$) is key, HPC and HTC are crucial.

B.1.1. Grif

Grif [265] code studies the speed of chemical reactions from the physical phenomena that rule the movement of atoms and molecules along the collision process. In this case, every simulation is on the range of seconds, requiring several millions for a single experiment.

Grif is devoted to simulate a chemical reaction of outmost importance for the life in Earth. It studies a chemical reactions happening in the atmosphere, $OH + CO \rightarrow H + CO_2$, which is the most important reaction on the removal of CO and OH from the atmosphere. It removes 90 % of the CO and 70 % of the OH and, as a result, about 20 % of the CO_2 of the atmosphere is produced. It also heavily affects other atmospheric species, with OH being critical in many other reactions.

Grif code studies the speed of chemical reactions from the physical phenomena that rule the movement of atoms and molecules along the collision process by employing Schrödinger's equation. It is specially suitable for grid environments as it has a high demand of CPU resources, does not have many memory requirements, and the data transfer is kept to a minimum.

B.2. Evolutionary Biology

The determination of the evolution history of different species is nowadays one of the most exciting challenges that are currently emerging in Computational Biology. The following three tools that are customary used by the scientific community have been adapted in this thesis to distributed environments.

B.2.1. jModelTest2

ModelTest (and its Java version jModelTest) [266] is a tool to carry out statistical selection of best-fit models of nucleotide substitution. It implements five different model selection strategies: hierarchical and dynamical likelihood ratio tests (hLRT and dLRT); Akaike and Bayesian information criteria (AIC and BIC); and a Decision Theory (DT) method. It also provides estimates of model selection uncertainty, parameter importances and model-averaged parameter estimates, including model-averaged tree topologies. jModelTest2 [267] includes HPC capabilities and additional features like new strategies for tree optimisation, model-averaged phylogenetic trees (both topology and branch length), heuristic filtering, and automatic logging of user activity.

jModelTest2, as well as its predecessors, makes use of PhyML [268], a phylogeny software based on the maximum-likelihood principle that implements new algorithms to search the space of tree topologies with user-defined intensity.

The calculations made with jModelTest2 can demand vast computational resources, especially in terms of processing power, so three parallel algorithms for model selection are available: a multithreaded implementation for shared memory architectures; a message-passing implementation for distributed memory architectures, such as clusters; and, a hybrid shared/distributed memory implementation for clusters of multicore nodes, combining the workload distribution across cluster nodes with a multithreaded model optimisation within each node. The second version with DRMAA capabilities instead of MPI ones has been used in this thesis.

B.2.2. MrBayes and PhyloGrid

MrBayes [269] is a program for doing Bayesian phylogenetic analysis. The program uses Markov Chain Monte Carlo (MCMC) techniques to sample from the posterior probability distribution. By default, MrBayes uses Metropolis-coupling to accelerate convergence; specifically, three *heated* chains are run in parallel with each regular *cold* chain. The heated chains sample from distributions obtained by raising the posterior probability with some factor smaller than 1, resulting in flattening (*melting*) of the peaks in the landscape defined by the posterior distribution. At specified intervals, the parameter values (the locations in the landscape) are swapped between cold and heated chains, which makes it possible for the cold chain to escape local peaks. This comes at the cost of increased computational complexity, but can really help convergence for difficult problems.

By default, MrBayes runs two independent analyses in parallel and calculates convergence diagnostics on the fly. This helps the user determine when to stop the analysis.

PhyloGrid [238] is a workflow based on Taverna [153] that makes use of MrBayes. It is able to execute MrBayes on grid resources and to be profited from a web service

interface. With the framework, some scientific results has been achieved [237, 236, 235] in understanding the relationship and differences among some regional genotypes of the human papillomavirus (HPV), human immunodeficiency virus (HIV) and Dengue virus (DENV).

B.2.3. ProtTest3

ProtTest [270] is a bioinformatics tool for the selection of best-fit models of aminoacid replacement for the data at hand. ProtTest makes this selection by finding the model in the candidate list with the smallest Akaike and Bayesian information criteria (AIC, BIC) score or Decision Theory (DT) criterion. At the same time, ProtTest obtains model-averaged estimates of different parameters (including a model-averaged phylogenetic tree) and calculates their importance [271]. ProtTest differs from its nucleotide analogue jModelTest in that it does not include likelihood ratio tests, as not all models included in ProtTest are nested.

ProtTest is written in Java and also uses the program PhyML [268] for the maximum likelihood (ML) estimation of phylogenetic trees and model parameters. The current version of ProtTest (3.2) [272] includes 15 different rate matrices that result in 120 different models when we consider rate variation among sites (+I: invariable sites; +G: gamma-distributed rates) and the observed amino acid frequencies (+F).

From the computational point of view, ProtTest3 is the High Performance Computing enabled version of ProtTest that can be executed in parallel in multi-core desktops and clusters. It also presents several parallel strategies as distinct execution modes in order to make an efficient use of the different computer architectures that a user might encounter: a Java thread-based concurrence for shared memory architectures (e.g., a multi-core desktop computer or a multi-core cluster node); an message-passing (MPJ Express library) parallelism for distributed memory architectures (e.g., HPC clusters); and, a hybrid implementation MPJ-OpenMP to obtain maximum scalability in architectures with both shared and distributed memory (e.g., multicore HPC clusters).

B.3. High Energy Physics

High Energy Physics has made lots of advances in the near past. Most of them, as the very well-known experimental evidence of the Higgs boson, have been possible due to the development of grid infrastructures and federations. This distributed environment was mainly promoted by CERN in order to simulate, process, and analyse the measurements of the Large Hadron Collider. However, the platform can be profited to obtain results for other type of studies in the same field, for example the cosmic radiation.

B.3.1. Nagano

The flux of cosmic rays at the highest energies ($> 10^{18}$ eV) is very small, and therefore, detectors with very large aperture have to be used. Among the different possible techniques, fluorescence telescopes arise as a very powerful alternative. This technique however still presents some open problems, such as a limited accuracy due to technological and atmospherical factors [273]. This obliges to interpenetrate the

obtained data to extract the relevant information, a task far from being immediate. For this sake, Nagano [72] is devoted to simulate fluorescence emissions and the energy deposited by electrons inside an observation volume of the desired proportions. This allows a thorough interpretation of available experimental data on the fluorescence yield and other related parameters.

Thus, Nagano lies on the area of Astroparticle Physics, helping on the research of the origin and propagation of ultra-high-energy cosmic rays.

Nagano [72] is a MC application based on the experiments of Nagano et. al [274, 274]. The calculation for a single electron takes only a fraction of second, but for a real world use case, a complete simulation comprises several millions of electrons.

B.3.2. XMM-Newton SAS software

XMM-Newton is the most sensitive X-ray satellite ever built and the largest satellite ever launched by ESA. It has been operating as an open observatory [275] since the beginning of 2000, providing X-ray scientific data through three imaging cameras and two spectrometers, as well as visible and UV images through an optical telescope. The large amount of data collected by XMM-Newton is due to its unprecedented effective area in the X-ray domain in combination with the simultaneous operation of all its instruments. All the data taken by this satellite are kept in the XMM-Newton Science Archive (XSA).

The Scientific Analysis System (SAS) [276] is a software suite for the interactive analysis of all the XMM-Newton data, making possible the tailoring of the data reduction to the scientific goal. In addition it makes possible a re-calibration of the data whenever new calibration files have been released. Due to XMM-Newton operates as open observatory, SAS is freely available and a large number of people are still working to improve it, although it is considered a fully mature package.

The large amount of data available makes necessary to optimise the management of hardware resources to prevent a data processing slow down. In this context, grid technology offers the capability of managing not only the user queries retrieving data from the archive, but also the online processing of that data. The execution of the SAS software on a grid has been successfully studied in [277]. However, the deployment of the new versions of the SAS software in grid infrastructures is not trivial, and requires an important effort from the VO administrators. This problem can be easily solved by storing virtual machine images at cloud repositories or making use of the contextualisation features to distribute the latest release. These approaches are the ones tackled in this thesis.

B.4. Matter Interactions

The effects of the radiation on either human tissues or materials are of great importance, mainly if radiation is able to produce ionization. In this sense, the target can be simulated by water molecules or solid lattices respectively. The described system presents multiscale components as fluid or molecular dynamics that must be simulated both in space and time. Such a numerical simulation can only be approached via HPC and HTC due to the great amount of particles to be taken into account and the physical laws to be calculated per step.

B.4.1. BEAMnrc

BEAMnrc [71] is a system for modelling radiotherapy sources, devoted to work on 3D treatment planning for radiotherapy, that is, it is devoted to simulate the radiation beams from radiotherapy units.

BEAMnrc is a general purpose Monte Carlo code that can simulate high-energy electron and positron beams, ^{60}Co beams and ortho-voltage units handling a variety of geometries entities that can even be put together by the user. Since the outcome of Monte Carlo simulations is based on random sampling, typically $\sim 10^8$ particle histories are needed for good accuracy, taking weeks of computation on a $\sim$ 2GHz processor. This remains the main hindrance in clinical implementation of Monte Carlo simulations.

Selection of input parameters can be very complex. In this thesis, we have used real examples $> 10^8$ particles and rectangular geometry, which represents use cases that can be approached in a hospital environment. Basically, two main factors must be considered: the lowest total energy for the production of secondary electrons and the desired residual range to the lowest energy for which an electron is transported. Both parameters usually range from 0.5 to 0.7 MeV and have a direct influence on the computing time, which can vary by even a factor of three.

B.4.2. FLUKA

FLUKA [278] is a general purpose tool for calculations of particle transport and interactions with matter. FLUKA can simulate with high accuracy the interaction and propagation in matter of about 60 different particles, including photons and electrons from 1 KeV to thousands of TeV, neutrinos, muons of any energy, hadrons of energies up to 20 TeV (up to 10 PeV by linking FLUKA with the DPMJET code) and all the corresponding antiparticles, neutrons down to thermal energies, and heavy ions. The program can also transport polarised photons (synchrotron radiation) and optical photons. Time evolution and tracking of emitted radiation from unstable residual nuclei can be performed online.

FLUKA can handle even very complex geometries, using an improved version of the well-known Combinatorial Geometry (CG) package. The FLUKA CG has been designed to track correctly also charged particles (even in the presence of magnetic or electric fields).

For most applications, no programming is required from the user. However, a number of user interface routines (in Fortran 77) are available for users with special requirements. In this thesis, up to a million particles have been simulated, each taking less than a second, for making studies on radiation interaction with matter. Materials can be simple elements or compounds, where an element can have either natural composition or consist of a single nuclide; and compound indicates a chemical compound or a mixture or an alloy (or an isotopic mixture) of known composition. An element can be either predefined defined giving its atomic number, atomic weight, density, name, and a material identification number.

B.4.3. GAMOS

The GAMOS [74] framework is based on GEANT4 [279] and is specialised in the simulation of the radiation with the body, i.e. medical applications in both fields of medical image (PET/SPECT) and radiation therapy (teletherapy and brachytherapy).

It also can simulate the needed doses to calibrate medical apparatus.

GAMOS offers a simple user interface covering the most common needs of a medical application, so that simulations with GEANT4 can be carried out without having to code in C++. At the same time, if the user has a special request, it permits to add new functionalities in a simple way, thanks to the utilization of the technology of plugins. GAMOS is compatible with the so-extended medical image format DICOM, so the calculation of doses in patients is a straightforward calculus. It also comes with an extensive set of tools that allows the user to get a detailed understanding of the simulation with a minimal effort.

B.5. Nuclear Fusion

Fusion energy aim is to become a commercial and viable source of energy by means of a clean, secure and long-term energy supply. Its work regime raise up to temperatures of $T \simeq 40$ KeV and, virtually, all of the atoms present in the gas are in an ionised stage, with electrons becoming separated from their nuclei. The resulting ions and electrons then form two intermixed fluids that become in an ionised gas, which remains almost neutral throughout, and it is called a plasma. Magnetic confinement is a promising possibility to contain materials at this plasma state, with two main confinement devices designs: stellarators and tokamaks. The research presented in this thesis is focused on the stellarator existing in the National Fusion Laboratory of Spain: the flexible Heliac TJ-II.

Plasmas are complex systems with a lot of non-linear processes and where temporal and spatial scales are represented by self-similarities and self-organization processes. The research in Plasma Theory can therefore help to advance in Chaos and Self-organization Theory. And vice versa, this kind of disciplines can provide tools for Plasma Physics and Fusion research. Thus, beyond the obvious interest that Plasma Physics has for commercial fusion development, it is connected with some of the present challenging disciplines in Physics such as Statistical Physics, Thermodynamics, or Fluid and Kinetic Theories. All these disciplines have still open problems whose solutions can help in commercial fusion achievement. Due to the complex nature of these open problems, computing is a key element for solving them. It could be mentioned for example that one millisecond plasma simulation will be performed within several days on a 50-100 TFlops machine like Mare Nostrum [280]. In 2014, one transport-time-scale simulation (i.e. one second in the energy confinement time scale) will take several weeks on a 4 PFlops machine, assuming that computer power doubles every 18 months.

This thesis is focused on the Neoclassical transport calculations for Nuclear Fusion, i.e. transport derived from the particle collisions and from the magnetic field inhomogeneities, which is always present in the fusion reactors. Its study can be done in several ways and can lead to know, for example, the efficiency for confining of a certain coil configuration. For the TJ-II stellarator case, it has been mainly calculated by means of two approaches: MC methods [281, 282] and Drift Kinetic Equation (DKE) solvers [244]. MC approaches only can offer an estimation of the diagonal part of the transport matrix. On the other hand, DKE solvers provide correct quantitative results of the complete transport matrix, with the drawback of high computation time and memory consumption. Both approaches are studied in this thesis.

B.5.1. DKES and DKEsG

The physical problem is to obtain the lowest energy flux to the plasma wall, so the transport produced must be obtained. The Neoclassical transport approximates the phenomenon by taking into account in a three-dimensional space the collisions and the slow and static behaviour of the electric field, so the diffusive and anisotropic system is a slightly non-equilibrium one.

DKES code [244, 245, 248] has been developed under a variational principle for the linearised drift-kinetic Fokker-Planck equation, which describes the transport in three-dimensional toroidally confined plasmas.

The code solves the entropy production and the residual function by using truncated Fourier and Legendre series to represent the dependencies of the magnetic field on the two angle variables, which correspond to the straight magnetic field line flux coordinates (so the geometry of the device is included) and on the pitch angle for collisions, respectively. The monoenergetic diffusion coefficients that DKES calculates are related to the diffusion of heat and particles, the bootstrap current, and the resistivity enhancement. All of these coefficients depend on the energy, the collision frequency, and the radial electric field, so the code obtains a wide set of values of coefficients by running the code with different values of the latter.

However, the monoenergetic diffusion coefficients obtained by DKES do not allow calculating the Neoclassical transport fluxes. In this thesis, an application was implemented to obtain the final Neoclassical transport coefficients form the DKES outputs. Moreover, both applications have been orchestrated by means of a computational workflow called DKEsG. This framework allows researchers to perform extensive transport calculations on distributed environments such as grid.

B.5.2. FAFNER2, ISDEP and FastDEP

FAFNER2 [226] performs the modelling of fast neutral beam injection (NBI) into three-dimensional toroidal plasmas, and calculates the trajectory of the resultant fast ions until they get lost or are absorbed by the system. For this purpose it follows a MC approach which, depending on the desired precision and problem, requires between 10^3 and 10^5 short simulations or samples.

The ISDEP code [283] is a MC application that calculates the distribution function of a minority population of ions in a magnetised plasma. It solves the ion equations of motion taking into account the complex 3D structure of fusion devices, the confining electromagnetic field, and collisions with other plasma species. The Monte Carlo method used is based on the equivalence between the Fokker-Planck and Langevin equations. This allows ISDEP to run on distributed computing platforms without communication between nodes with almost linear scaling.

With FastDEP [73], the birth positions of the fast ions in the fusion plasma can be calculated and, lately, ion trajectories in the plasma background can be followed, i.e. FastDEP is a workflow that couples FAFNER2 and ISDEP. In general, a typical calculation is composed of several thousands of tasks that last from 1 to 3 CPU hours. As an additional particularity, the large requirements in terms of input data make FastDEP to employ storage elements to download the required data and to save results when runs in remote grid sites.

B.6. Solid State Physics

There is a plethora of problems related to Solid State Physics that only can be tackled with computational simulations. In this way, theory, experimental facilities, and HPC/HTC are essential in this area: accelerators, Lattice Field Theory, Lattice Gauge Theory, Molecular Dynamics, or Soft Condensed Matter Physics are some examples. In this way, methods like density functional or *ab initio* theories are the based on which the computing codes rely.

B.6.1. DiVoS

The vortex lattice dynamics close to critical temperatures is a topic of interes in the Superconducting field. Superconducting vortex lattice pinning and vortex lattice dynamics are strongly modified by arrays of nanodefects embedded in superconducting films [284]. Magnetoresistance measurements are a perfect tool to study these effects, since resistance versus applied magnetic fields shows deep minima when the vortex lattice matches the unit cell of the array, due to geometric matching occurs when the vortex density is an integer multiple of the pinning centre density. These phenomena are ruled by the balance among different interactions, (i) vortex-vortex, (ii) vortex-artificially induced pinning centre (array of nanodefects), (iii) vortex-intrinsic and random pinning centres. However, the large roughness of the sample surface precludes the use of standard local probe methods to detect experimentally the vortex position and symmetry of the vortex lattice.

The DiVoS code [228] explores the possibility to simulate the commensurability experiments in the framework of the Langevin equation of motion, but without any initial conditions neither constraints and using only as input the vortex-vortex interaction and the periodic pinning sites (array unit cell). Thus, this code can calculate different values and positions for different lattices in size, matching field values and geometry of the pinning sites, which allows having a picture of the different vortex lattices which develop for the main and upper order matching conditions as well as an estimated magnitude of their interactions. Additionally, the calculation can be performed on grids in a reasonable amount of time. As consequence, some new results have been achieved [229] for different vortex lattice configurations on periodic potentials.

Appendix C

Physics of Transport Codes and Physical Results

The magnetic field used to confine the plasma in stellarators is generated by external coils. The cost of fusion power depends on how efficient the magnetic configuration to confine the plasma pressure is, and on the fraction of power that must be used to maintain the plasma and the magnetic field, i.e. the recirculating power fraction. Then, to construct new stellarator devices as much efficient as possible, a rigorous study of the Neoclassical (NC) transport that occurs in the confined plasma is a key factor. The physical problem is to obtain the lowest energy flux to the plasma wall, so the transport must be estimated and understood in order to find strategies to reduce it. The NC Theory approximates the phenomenon by taking into account the three-dimensional structure of the magnetic field, the collisions and long-time average electric field, so this diffusive and anisotropic system is a slightly non-equilibrium one.

C.1. Introduction

The calculation of NC transport in three-dimensional stellarator systems is difficult because of the complex structure of the field. However, it can give us valuable insights into the transport properties. Analytical methods, when applied to real stellarators, seem to be impractical because of the large number of magnetic-field components that have to be retained in the calculations, and this problem is especially severe for the TJ-II stellarator [260] due to its very broad magnetic field spectrum. Numerically, NC transport in stellarators can be calculated by means of two methods: The Drift Kinetic Equation (DKE) solvers; and Monte Carlo methods. Meanwhile the latter only produce the diagonal part of the transport matrix, DKE solvers allow computing the whole matrix starting from the monoenergetic diffusion coefficients. Thus, if the bootstrap current and the parallel resistivity are required, DKE solvers must be used. Nevertheless, the numerical implementation of such codes scales unfavourably with the required number of magnetic-field modes (harmonics) and presents uncertainties in the calculations for the low collisionality regime. With the advent of more powerful computational architectures such as grids, the complete calculation can be executed in a reasonable time once the code has been adapted and optimised to be run on them.

First calculations of NC transport for the TJ-II stellarator [285] were made using the DKES code [244] just for one magnetic surface $r = a/2$. Several conclusions of interest can be found in that work and they demonstrated the feasibility and interest of using DKE solver on a flexible Heliac stellarator with a rich magnetic-field structure. These studies with the same magnetic surface at $r = a/2$ were complemented with more calculations [286] considering several configurations and plasma pressures. Using the DKES code and some analytical approximations [287] the bootstrap current and its effect on the rotational transform were also addressed. Both studies were performed before TJ-II started operation, so they were mainly devoted to obtain general dependencies with only zero-dimensional estimations. Thus, starting from their conclusions, Tribaldos [281] increased the study of Neoclassical transport in TJ-II by means of Monte Carlo techniques. Later, this work was complemented with the ISDEP code [283]. It is important to bear in mind in this point that sometimes it is necessary to introduce approximated expressions for NC coefficients, which is the case of the PROCTR [288] and PRETOR [289] stellarator codes. The PRETOR code, which was developed at JET for allowing the simulation of the radial variation and the temporal evolution of the main physical magnitudes of fusion plasmas, and its adoption to the stellarator case has been also used [290, 291] for similar studies in TJ-II. These results were compared with the previously obtained ones using the PROCTR code. In addition, the reader can also find calculations that could be applied for TJ-II by means of guiding centre and bounce-averaged MC simulations [282] and numerical solutions of the linearised Drift Kinetic Equation [292]. These methods can offer accurate results depending on the numerical considerations that are taken into account when calculated, i.e. the considered approximations to make the problem tractable.

To the date, DKE solvers have been usually executed on shared memory systems. Examples can be found in the work from Ogawa *et al.* [293], where the Neoclassical transport in the banana regime has been analysed with the DKES code [244] for the Large Helical Device, or in the calculation of the plasma local diffusion coefficients [294]. Lately, flux-surface averages and plots of the two-dimensional structure of the ion flow velocity for different stellarator configurations, such as that of QPS, have been obtained by means of the Moments Method [245, 248]. This method calculates the components of the above flow velocity that are necessary for its determination, i.e. the average flux-surface (weighted magnetic field), parallel flow velocity and the ambipolar electric field. These are obtained by using the Neoclassical transport matrix from [295] that relates the Neoclassical radial fluxes of particles and heat to the gradients of density, temperature, and potential for electrons and ions. The elements of this transport matrix are written in terms of energy integrals over the three monoenergetic transport coefficients obtained by running the DKES code. Nevertheless, this DKES version proposed by Spong, in order to better address the physics of particle drifts at low collisionality, is extended first to 4 dimensions (three spatial ones and one in the velocity space) and eventually 5 dimensions (three spatial ones and two in the velocity space). Thus, it utilises OpenMP parallelization and maps the 3D model across multiple symmetric multi-processor (SMP) nodes, i.e., each node contains a diagonal block matrix that looks like a separate 3D model. This calculation is then appropriate for very dense nodes of tightly coupled processors and, at the same time, is not suitable for grid infrastructures, so the old van Rij & Hirshman execution mode has been used on distributed platforms.

As a conclusion, all these previous works have shown the necessity and interest

of having at researchers'disposal a tool for performing extensive NC transport calculations. Since Grid Computing can offer solutions for many of the computationally intensive problems coming from the complex simulation of fusion devices, one aim of this work is then to offer a new optimised version of DKE solvers. To do so, it is important to point out that the parametric and sequential nature of the problem makes possible its division in minimal tasks that can run on clusters and grid environments, so the use of shared memory computers can be avoided. The adaptation process to the grid of the variational releases of the Drift Kinetic Equation solver code [244, 245, 248], the implementation of a new module to calculate the transport coefficients, and a workflow who efficiently and automatically rules the execution of jobs and their dependences, resulted in the Drift Kinetic Equation solver for Grid (DKEsG) framework described in Section 4.4. It has been possible to compare DKE with Monte Carlo approaches with this new tool and the adaptation of ISDEP, the performance of which have been improved using the GWpilot framework (see Chapters 9 and 10). For this purpose, plasma fluxes have been calculated from the outputs of both approaches for the TJ-II device. Moreover, the relationship between NC transport and rotational transform has been evaluated calculating the effective ripple with DKEsG and GWpilot.

C.2. Flux calculation

Although turbulent transport can be dominant in some specific regimes and plasma zones, there is always a background transport provoked by collisions, electric field and inhomogeneity of the background magnetic field. This is called Neoclassical (NC) transport and is present both in tokamaks and stellarators, although it is larger in the latter. The NC transport is driven by the gradients of thermodynamic quantities, and its computation is based on the so-called Neoclassical ordering, which implies a small radial orbit size, a conservation of kinetic energy and an infinite parallel transport along field lines. Moreover, NC calculations assume that the transport has a diffusive nature and that the vector of fluxes is given by the product of the matrix of transport coefficients times the vector of gradients. Even considering these assumptions, the NC calculations require a huge computation effort.

NC transport is calculated by solving the Drift Kinetic Equation (DKE), which is the kinetic master equation for magnetically confined plasmas. The DKE Solver code (DKES) is customarily used by the fusion community [244] to obtain the so-called mono-energetic transport coefficients, which are then convoluted with the plasma velocity distribution function. This code solves the linearised drift-kinetic Fokker-Planck equation by using a variational principle, which describes the transport in three-dimensional toroidally confined plasmas. In addition, the effect of magnetic drifts on the orbits of the perturbed distribution function is neglected, so the study of resonant superbanana orbits is precluded. These drifts have small values for toroidal devices with large aspect ratio $A = R/a$ (being R the major radius and a the minor one), but could be significant at very low collision frequencies for radial electric fields satisfying $e\Phi/T > A^{-1}$. Therefore the problem can be approached by reducing the drift-kinetic equation phase-space from five to three dimensions while leaving its conservative and symmetry properties invariable.

Thus, these properties are used to build a variational function that represents the entropy production, i.e. a maximizing and minimizing principle for the entropy production rate which also implies upper and lower bounds on the transport coef-

ficients. The code solves the entropy production and the residual function by using truncated Fourier and Legendre series to represent the dependencies on the two angle variables which correspond to the straight magnetic field line flux coordinates (so the geometry of the device is included) and on the pitch angle, respectively. The monoenergetic diffusion coefficients that DKES calculates are related to the diffusion of heat and particles, the bootstrap current and the resistivity enhancement. All these coefficients depend on the energy, the collision frequency and the radial electric field, so the code obtains a wide set of values of coefficients by running the code with different values of the latest.

However, this code assumes that the typical orbits of the particles are radially narrow enough to neglect the plasma variations, so that particle transport depends only on the local characteristics of the plasma. Nevertheless, there are cases in which this condition is not fulfilled. Thus, a Monte Carlo code, such as ISDEP [283], can be used because it does not require any assumption on either orbit widths or energy conservation or on the diffusive nature of transport. In the case of the TJ-II stellarator, a violation of NC ordering in the low collisionality regime is expected [296]. Therefore, the comparison of the results of DKES with a global transport code such as ISDEP is necessary to assure the validity of the assumed ordering in such a regime.

The comparison of the results given by ISDEP and DKES implies the calculation of fluxes rather than of transport coefficients because ISDEP provide the fluxes without the assumption of diffusive transport. Thus, it will be necessary to estimate the NC fluxes to be compared with the ones given by ISDEP. The estimation of the fluxes requires that consideration be given to the plasma profiles, from which the gradients of density, temperature and electric potential must be estimated. These profiles are taken from the TJ-II experimental results [283].

The NC fluxes are given by the following equation:

$$\begin{pmatrix} \Gamma \\ Q \\ j \end{pmatrix} = L \begin{pmatrix} \frac{n'}{n} + \frac{3}{2} - \frac{eE}{T} \\ \frac{T'}{T} \\ \frac{-e}{T} \frac{\langle \vec{E}\,\vec{B} \rangle}{\langle B^2 \rangle} \end{pmatrix} \tag{C.1}$$

where the left hand side is the vector of radial fluxes of particles, heat and charge average over the magnetic surface (top to bottom). Prime denotes radial derivative, n is the plasma density, T is the temperature, E is the electric field, and B is the magnetic field. These fluxes are given in terms of the thermodynamic force vector and the symmetric matrix of transport coefficients L. The calculation implies one equation for every different species of the plasma.

C.2.1. Determination of fluxes from NC transport

The calculation of NC transport is a basic task in stellarators and three-dimensional systems in general but is considered one-dimensional because it is assumed that the transport parallel to the magnetic field is infinite, and therefore, only the fluxes perpendicular to the surfaces generated by the magnetic field lines make sense.

The fluxes given in Equation C.1 can be written as:

$$\Gamma_i = \sum_{j=1,3} L_{ij} A_j \tag{C.2}$$

where A_j are the thermodynamic forces that appear in Equation C.1, and the transport coefficients for every species present in the plasma (electrons, ions and impurities) are given by:

$$L_{ij} = \frac{2n}{\pi} \int_0^\infty dK \sqrt{K} e^{-K} g_i(K) g_j(K) D_{ij}(K) \tag{C.3}$$

where $g_1 = g_3 = 1$, $g_2 = K = mv^2/2T = (v/v_T)^2$, with v being the particle speed, K the kinetic energy normalised to the temperature and D_{ij} the so called monoenergetic diffusion coefficients. L is a symmetric matrix ($L_{ij} = L_{ji}$), so only six independent coefficients appear. These i, j indexes correspond to the ones of the transport matrix in [244], where $i = 1$ coincides with the particle transport, while $i = 2$ is related to heat transport and $i = 3$ corresponds to charge transport. The mixed $i \neq j$ indexes of the transport matrix represent the contribution to the fluxes of the gradients of other thermodynamic forces. The monoenergetic coefficients are estimated by DKEsG-Mono separately as functions of collisionality, electric field and the kinetic energy of the particle:

$$D_{11} = D_{12} = D_{21} = D_{22} = -\frac{1}{2} v_T \left[\frac{B v_T}{\Omega} \left(\frac{dp}{dr} \right)^{-1} \right]^{-2} K^{\frac{3}{2}} \hat{D}_{11} \tag{C.4}$$

for the diffusion of heat and particles (related to temperature T and density n),

$$D_{31} = D_{32} = -D_{13} = -D_{23} = -\frac{1}{2} v_T \left[\frac{B v_T}{\Omega} \left(\frac{dp}{dr} \right)^{-1} \right] K \hat{D}_{13} \tag{C.5}$$

for the bootstrap current and thermo-diffusion,

$$D_{33} = -\frac{1}{2} v_T K^{\frac{1}{2}} \hat{D}_{33} \tag{C.6}$$

for the resistivity enhancement.

In these expressions, the dependence of $\hat{D}_{ij}$ on the kinetic energy K, the collision frequency ν and the radial electric field E is determined by just two parameters:

$$\hat{D}_{ij} = \hat{D}_{ij} \left[\frac{\nu(K)}{v_T \sqrt{K}}, \frac{E_s}{v_T \sqrt{K}} \right] \tag{C.7}$$

with v_T being the thermal velocity of the considered species. The two variables that appear in Equation C.7 are the above ones referred to as $CMUL = \nu(K)/v_T\sqrt{K}$ and $EFIELD = E_s/v_T\sqrt{K}$. Because K plays the parametric role in the resolution of Equation C.3, a wide variation of these two input parameters is necessary for obtaining accurate NC transport coefficients, and it is valuable to build an extensive database. Such a database would include parameters for a large variety of particle energies, collisionalities, electric fields and radial positions and could be filled in for a variety of magnetic configurations and devices. Additionally, the determination of a vast database compiling monoenergetic and transport coefficients is very useful for coupling DKES to a transport evolution code, which can use those values as input data.

With T_i measured in KeV and M_i measured in atomic mass units,

$$v_T = 4{,}392825 \times 10^5 \left[\frac{T_i}{M_i} \right]^{\frac{1}{2}} \tag{C.8}$$

$$\frac{Bv_T}{\Omega} = 4{,}552878 \times 10^{-3} \frac{[T_i M_i]^{\frac{1}{2}}}{Z_i} \tag{C.9}$$

where $Z_i = e_i/e$ is the ion charge.
Substitution from C.8 and C.9 in Equation C.3 gives:

$$L_{ij} = -4{,}552878 \, [T_i]^{\frac{3}{2}} \, [M_i]^{\frac{1}{2}} \left(Z_i \frac{dp}{dr} \right)^{-2} \times \left[\frac{2}{\sqrt{\pi}} \int_0^{\infty} K^{i+j} e^{-K} \hat{D}_{ij} dK \right] \tag{C.10}$$

What the original DKES code [244] performs is the calculation of a wide set of values of $\hat{D}_{ij}$ by running with different values of K, ν and E_s as input data. Equivalently, the electrostatic potential ϕ can be provided. Then, an analytical function can be fit for

$$L_{ij} = L_{ij} \, (v, \phi) \tag{C.11}$$

and a Maxwellian distribution in terms of

$$L_{ij} = L_{ij} \, (n, T, \phi) \tag{C.12}$$

is obtained; in this way the transport coefficients L_{ij} can be calculated as functions of the plasma parameters. Of course, they can also be written directly as functions of the collisionality and the electric field. This latter part is the one that has been additionally implemented in this work. The integration in Equation C.3 is performed with a chosen quadrature routine together with interpolation of the function.

The way that these calculations can be performed depends on the physical regime under study, i.e. the collision frequency value in the plasma. Depending on it -the high (low) number of collisions represents a lower (higher) plasma temperature or a higher (lower) plasma density- the user must select the number of Fourier and Legendre polynomials and provide physical parameters of the plasma regime to reproduce the system, so the computational load is directly affected.

C.2.2. The Monte Carlo approach

The use of ISDEP was motivated by the violation of NC ordering for low collisionality ion transport in TJ-II, as has been shown in [283] and [296]. In these plasmas, the radial width of orbits is large in comparison with the typical plasma lengths, and the electric field is strong enough to violate the conservation of ion kinetic energy. Due to these facts, collisional transport is not diffusive even when no turbulence is included in the calculations; thus, the fluxes cannot be written in terms of transport coefficients as in Equation C.1. To compute the fluxes in such conditions, the Monte Carlo method must be used to calculate the trajectories of a huge number of particles (typically 10^6) in the 3D geometry of the devices. Therefore, particles suffer collisions with the background plasma and the effect of the electric field. Additionally, the particles are launched following the experimental profiles, and their trajectories are estimated using the guiding centre approximation, i.e., ignoring the fast gyro motion around the magnetic field lines. These techniques are not only used in TJ-II or similar complex devices but they must also be used in tokamaks when the collisionality is so low that ions are in the so-called banana regime and the typical orbit widths are large. This is the case for the thermal ion transport in the ITER device [297].

C.3. The effective ripple

Hinton and Hazeltine [298], and more recently Belli and Candy [299], obtained the NC transport coefficients semi-analytically for a large aspect ratio tokamak, showing an explicit dependence on the rotational transform ($\iota/2\pi$), which drives one to believe that this parameter has influence on the NC properties of the device. Thus, in a tokamak, the NC transport coefficients vary with $\iota/2\pi$ for a given effective collisionality as:

$$D_{ij} \propto q^2 = (\iota/2\pi)^{-2} \tag{C.13}$$

This predicts an improvement of the confinement with the inverse of the safety factor (q). Therefore, NC transport and turbulent one diminish with the rotational transform in tokamaks.

The main configuration ingredient that originates NC transport is the magnetic ripple. In an axisymmetric tokamak the ripple comes from the poloidal variation of the magnetic field, therefore it is related to the rotational transform. In a stellarator no such a clear dependence can be extracted, since there is more freedom to design the magnetic configuration. In any case, the NC transport properties of a configuration are summarised in a single parameter, the effective ripple [300], given by:

$$\varepsilon_{eff} = \lim_{\nu^* \to 0} \left(2 \left[\left(\frac{3\pi}{4} \right)^{1/2} \nu^* \hat{D}_{11} \right] \right) \tag{C.14}$$

Where ν^* is the effective collisionality:

$$\nu^* = \frac{qR\nu(K)}{v_T} = \frac{R\nu(K)}{\iota v_T \sqrt{K}} = \frac{R}{\iota} CMUL \tag{C.15}$$

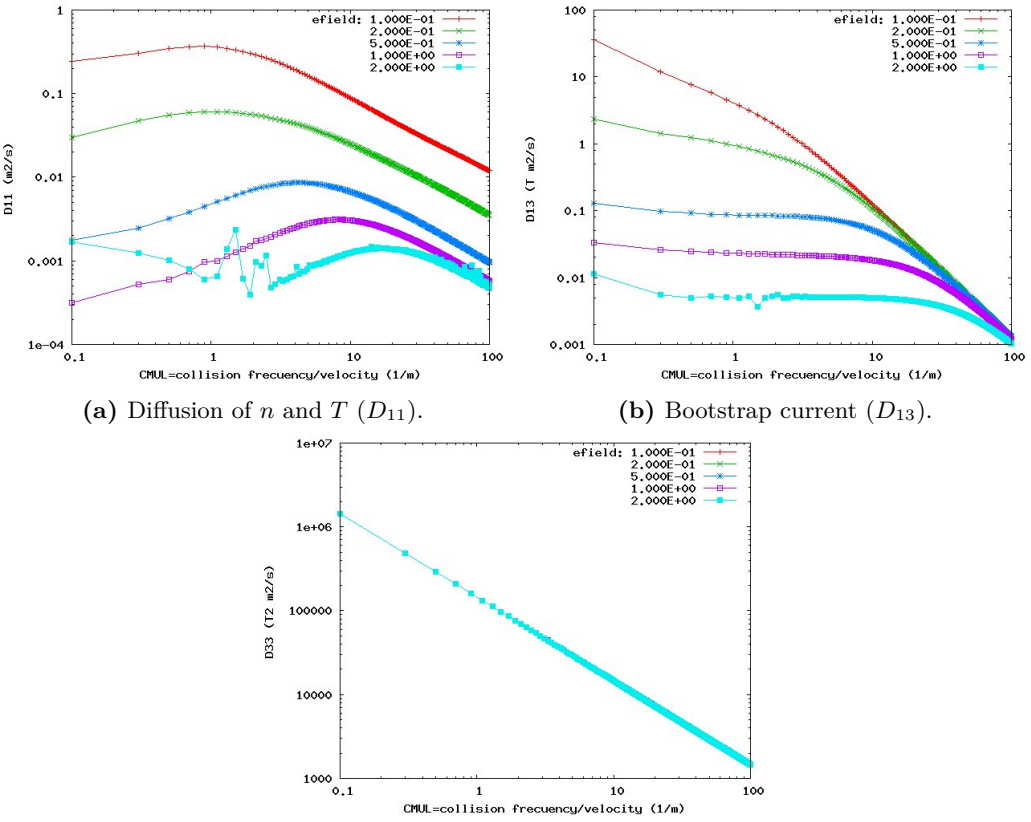
being R the major radius.

To check the possible relation between the $\iota/2\pi$ scaling and the NC transport in stellarators, the effective ripple must be calculated because it provides a clear idea of the quality of the magnetic configuration for NC confinement, without the necessity of exploring different collisional regimes.

C.4. A summary of the results

When running DKEsG-Mono, a large number of tests varying the number of Fourier and Legendre polynomials must be performed in order to find the optimal ratio between computational time and accurate physical results for every concrete study. However, how well the Fourier-Legendre series approximate the exact distributions is determined by an analysis of the convergence of the diffusion coefficients as the number of polynomials is varied, so this factor is more critical. Besides, the variation of this ratio is not totally free since it must deal with the geometry of the device to be simulated, the heliac stellarator TJ-II in the case of the results presented through this section.

Even more, not only real values for the geometry of the TJ-II (represented with their Fourier coefficients and radius of the magnetic surface) have been selected, but also some other parameters.



(a) Diffusion of n and T (D_{11}).

(b) Bootstrap current (D_{13}).

(c) Resistivity enhancement (D_{33}).

Figure C.1: Evolution of the diffusion coefficients as a function of collisionality (represented as CMUL) for different values of the electric field ($e\Phi/T$).

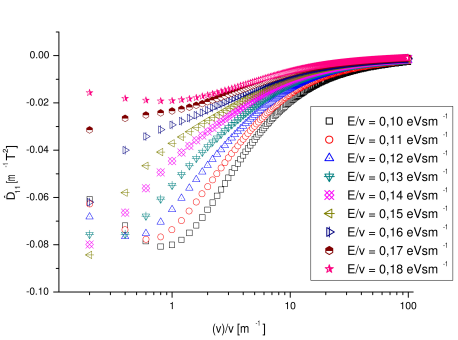
C.4.1. First plasma results from DKEsG

Thus, a test for average plasma minor radius $r = 4$ cm ($\rho = 0.021$) has been performed. Other magnitudes such as the temperature, the atomic mass and the potential have all been set to the unity, i.e. T, n, ϕ variables in Equation C.10. Finally, a selected set of 100 Legendre polynomials and 343 Fourier harmonics has been used to represent the TJ-II configuration and the distribution function. This approximation ensures reliable physical results.

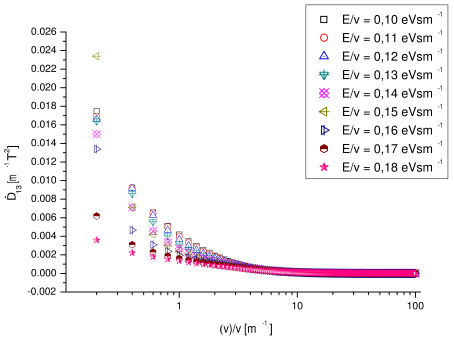
C.4.1.1. Monoenergetic diffusion coefficients

Figure C.1 is the plot of the three independent monoenergetic coefficients (D_{11}, D_{13} and D_{33}) as a function of collisionality (represented as the input parameter CMUL) for different values of the electric field. These values correspond to the ones obtained from the calculation described in Subsection 4.5.2.

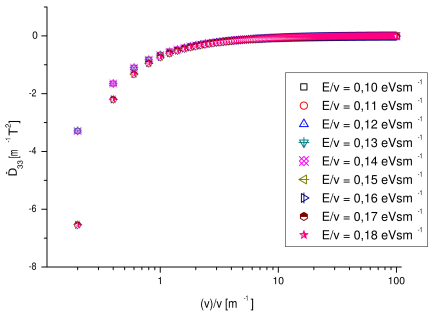
In the plots, different convergence behaviour can be seen; this is indicative of the suitability that the Fourier-Legendre representation has with respect to the experiment carried out, namely, if the near constancy of the distribution along a field line in the low-collision-frequency regime is well represented or not [244]. In addition to this,



(a) Normalised ($\hat{D}_{11}$) diffusion of n and T.



(b) Normalised bootstrap current ($\hat{D}_{13}$).



(c) Normalised resistivity enhancement ($\hat{D}_{33}$).

Figure C.2: $\hat{D}_{ij}$ normalised diffusion coefficients calculated by DKEsG-Mono in the performed test. They are independent of T, and n, but proportional to the $\hat{D}_{ij}$ ones.

there is a similar evolution of the diffusion coefficients as a function of collisionality between our results and those provided in [244] for common axis scales and electric field values. This fact is of outmost importance as it has been previously mentioned about the goal of this work, i.e. provide new DKES executions on grids in a better computational performance way, but physically valid accurate too.

To obtain more accurate plasma results, a parameterization with a lower increment in the collisionality values (horizontal axis in Figure C.1), i.e. lower granularity, can be considered. However, the current scan of CMUL values will guarantee the necessary precision to allow solving the Equation C.3, as it is performed in next subsection. Moreover, if the CMUL interval is carefully selected, ~ 60 values are enough to just achieve minimal significant results, as have been demonstrated in Subsection C.4.2.

C.4.1.2. NC transport coefficients

To automatically solve Equation C.3, DKEsG framework should be run in workflow mode as has been performed in Subsection 4.5.3. With the results obtained, Figure C.2 is plotted. In contrast to previous figures, Figure C.2 shows the three independent coefficients $\hat{D}_{ij}$ that DKEsG-Mono has calculated on the grid as functions of collisionality for different values of the electric field. These parameters are conver-

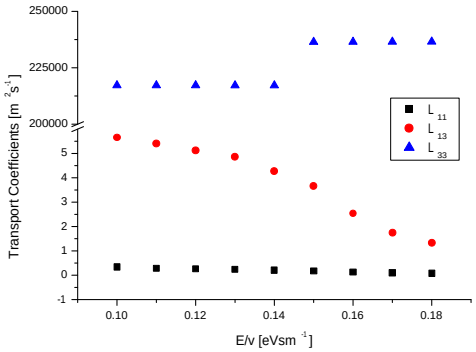


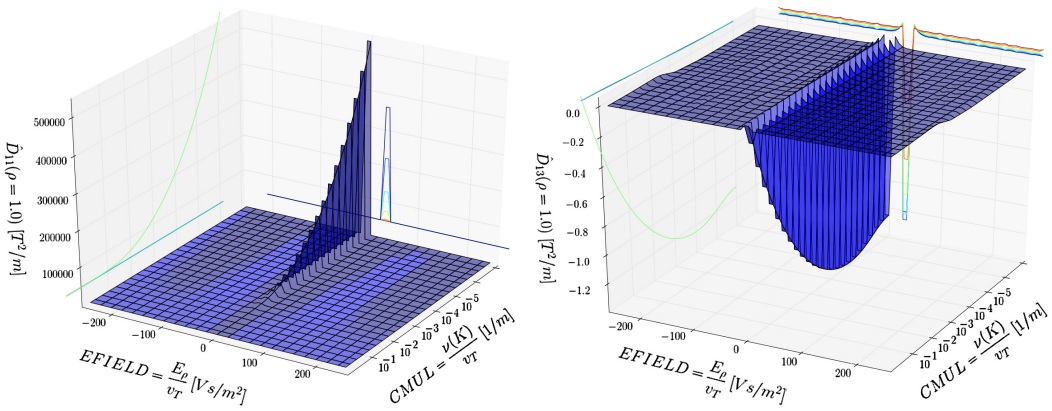
Figure C.3: Neoclassic transport coefficients L_{ij} calculated by the DKEsG framework.

ted into the monoenergetic coefficients D_{ij} by multiplying by constants associated to the particular coefficient and dependent on plasma state.

Figure C.3 shows the NC transport coefficients calculated by DKEsG-Neo module from previous ones. It can be seen how the monoenergetic transport coefficients vary with the collisionality for different values of the electric field. As it is usual in this theory, it is seen that the electric field tends to reduce the transport and to improve the confinement. The behaviour with respect to the collisionality is much more complex, depending on the collisionality regime (long or short mean free path, LMFP or SMFP) and on the coefficient under study. These results change drastically when studying different confinement devices or configurations and will be of importance in next subsection.

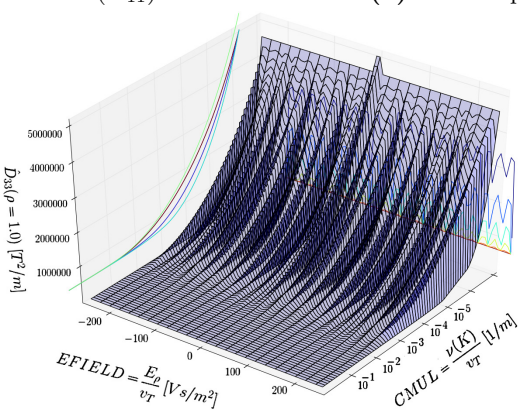
It has been shown in previous works that the dependence of NC transport on the TJ-II magnetic configuration is small, provided that the size of the configuration is similar, so in this work only the so-called standard configuration of TJ-II has been considered. It is important to bear in mind that the suitability that the Fourier-Legendre representation of the distribution function refers to the near constancy of the distribution along a field line in the low-collisionality regime, so it can be inferred if it is well represented or not [244]. For the sake of comparison, a similar evolution of the diffusion coefficients as functions of collisionality between the results of this work and those provided in [244] for common axis scales and electric field values can be found.

Some results related to the transport coefficients are also shown in Figure C.3. As it was mentioned in the Subsection 4.5.3, integrations over the energy space have been also carried out for obtaining them. The previous fittings for the diffusion coefficients which an analytical function inside Equation C.10 showed good quality for the regression coefficient not lower than 0.9991. As can be seen, a possible resonance phenomenon can be found for the L_{33} coefficient. This figure also shows that the influence of the electric field in the average is very small for particle transport, which is an effect to be attributed to the fact that the electric field effect changes drastically with the collisionality, thus compensating the variations of the coefficient when the average on the collisionality is performed. On the other hand it is possible to observe a reduction of heat transport when the field is increased together with an enhancement of the bootstrap current.



(a) Diffusion of n an T ($\hat{D}_{11}$)

(b) Bootstrap current ($\hat{D}_{13}$)

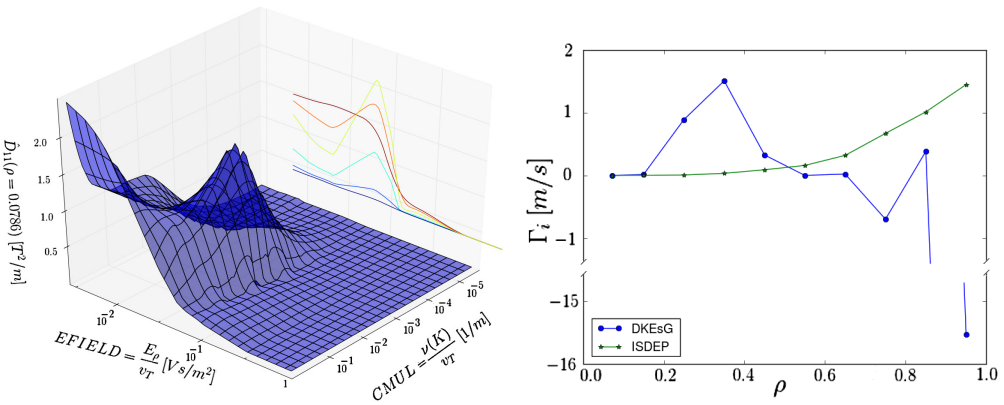


(c) Resistivity enhancement ($\hat{D}_{33}$)

Figure C.4: Normalised monoenergetic coefficients of the outer radial plasma position in TJ-II. To accomplish the representation of these surfaces, 3,672 DKEsG-Mono tasks and approximately 663 CPU hours from the grid have been consumed.

C.4.2. Comparison of fluxes

In some preliminary tests similar to the commented above, different parameter variations for a unique TJ-II configuration has been performed to determine the accuracy of the DKEsG results with respect to the fluxes obtained by ISDEP. In this sense, one important DKEsG-Mono parameter is the number of Legendre polynomials used to describe the distribution function, which must be large enough to represent the pitch angle scattering in the long mean free path (LMFP). It has been checked that the result converges for 200 polynomials even under the LMFP condition. Thus, the number of Legendre polynomials was fixed to 200 for the calculations devoted to the comparison. Additionally, the fluxes must be calculated for a set of radial positions (normalised as ρ parameter). Obviously, now the temperature and the atomic mass should take real values for the calculation. In this sense, a plasma composed only by protons is considered, and consequently $T_i = 100$ eV constant and $M_i = 1.007276\ u$ are selected.



(a) $\hat{D}_{11}(\rho = 0.0786)$. **(b)** Comparison of ion particle fluxes

Figure C.5: $\hat{D}_{11}$ representation of an inner radial plasma position ($\rho = 0.0786$) in TJ-II (figure (a), left) and final comparison of ion particle fluxes obtained by DKEsG and ISDEP (figure (b), right).

C.4.2.1. Determining the accuracy of coefficients

The experiments performed in Subsection 9.5.2 are motivated by the needed of verifying the accuracy of DKEsG-Mono calculations for the comparison of fluxes. The three independent monoenergetic coefficients obtained are shown in Figure C.4 and illustrate the necessity of performing a dense parameter scan for correctly analysing the data. As expected, the plotted functions are even in the electric field, and larger uncertainties appear in the LMFP regime. Note that the axis in Figure C.5-(a) does not exactly represent the $CMUL$ and $EFIELD$ values but ν/v_T and E/v_T, respectively, instead.

It is seen in Figure C.4-(a) that the particle diffusivity quickly increases for low values of the electric field, showing that this quantity has a strong influence on NC transport. It is seen also that this coefficient strongly increases as collisionality decreases, which is an intrinsic property of NC transport. Figure C.4-(b) shows that the bootstrap current tends to zero as the collisionality is increased and that the collisionless asymptotic value is recovered [301]. Figure C.4-(c) shows the dependence of the conductivity on both the electric field and the collisionality: it shows a very weak dependence on electric field and increases strongly for low values of the collisionality, as could be expected considering that the driven current is larger for low collisionality.

The coefficients shown in the figures are symmetric with respect to the Y axis, which corresponds to the electric field. Note also that the three coefficients $\hat{D}_{ij}$ obtained from DKEsG-Mono executions have different parity (positive or negative); a fact that is related to the bound limits for the convergence, i.e., their difference represents the error bar of the calculation. This is important and is directly stored for later use, as a source for the calculation of the resulting normalised L_{ij} and their propagated errors. In this sense, despite the whole Legendre polynomials set, the error at low collisionalities continues being too high to ensure an accurate calculation of L_{ij}. Thus, the main computation must be calculated with both $EFIELD$ and $CMUL$ values fitted to a logarithmic scale into a range closer to zero.

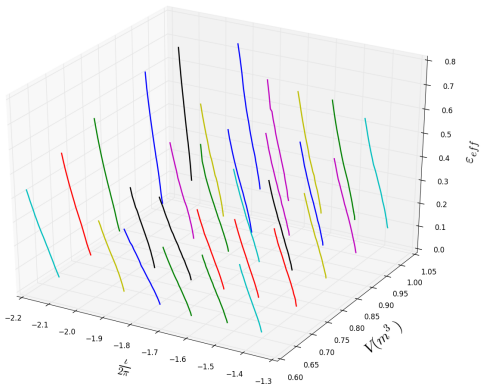


Figure C.6: Representation of the effective ripple (ε_{eff}), as a function of volume and rotational transform in TJ-II. The lines show the different values at different radial positions, where different values of the rotational transform appear.

C.4.2.2. Comparison between ISDEP and DKEsG

The comparison between the fluxes calculated from the results of DKEsG and ISDEP allow evaluating what extend NC ordering can be applied in TJ-II plasmas as well as its influence.

To obtain ISDEP outputs and the change in the parameter scan of $EFIELD$ and $CMUL$ justify the experiments performed in Subsection 9.5.3. Now, the particle diffusion coefficient ($\hat{D}_{11}$) estimated with DKEsG-Mono is presented in Figure C.5-(a) in a correct range of collisionalities and electric fields that successfully allows the integration of Equation C.3 to obtain the transport coefficient L_{11} and subsequently the particle flux. The integration in K of Equation C.3 is solved assuming that both $CMUL$ and $EFIELD$ parameters decrease monotonically as K increases. Then, the previously calculated $\hat{D}_{ij}$ selected for every K depend on the ν and E values initially considered, thus varying the obtained L_{ij} values. These ν and E values are based on the experimental profiles of TJ-II. Their profiles are calculated from the experimental data and can also be seen in [283]. As a consequence, there is a continuous reutilisation of the data included in the database, which must be previously selected and submitted to the grid as an input file together with DKEsG-Neo.

The ion fluxes normalised by the particle density, calculated by DKEsG (assuming the NC ordering) and ISDEP for the same density, electric field and temperature profiles, are shown in Figure C.5-(b) as a function of the normalised radius, which is the main physical result of this subsection. Important differences can be appreciated in the two flux profiles. The ion flux estimated by ISDEP is monotonic and increases with the radius. The maximum value of the flux is located at outer radial positions because the magnetic ripple is high at those positions, thus enhancing the particle transport. In contrast, the flux estimated by DKEsG is non-monotonic and presents a maximum around the radial position of 0.3, which is due to the combination of the values of the magnetic ripple, the collisionality and the electric field. Interestingly, this flux presents negative values for radial positions about $r/a = 0.77$. This means that the particle

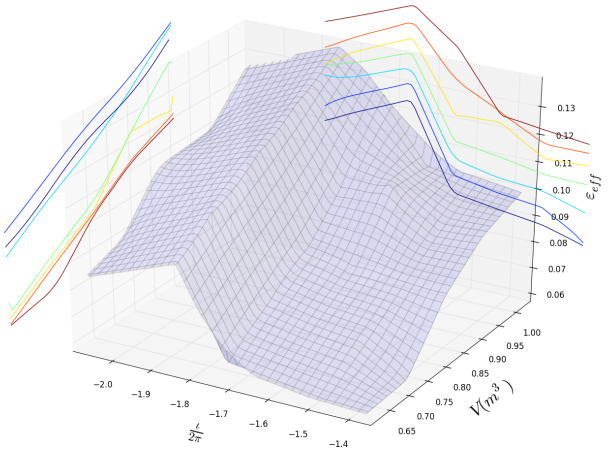


Figure C.7: Representation of the effective ripple (ε_{eff}), as a function of volume and rotational transform in TJ-II for $\rho = 0.007$. The lines extract the dependence of the effective ripple with $\iota/2\pi$ for constant volume and with volume for constant $\iota/2\pi$.

flux is inwards there, which is astonishing but can be explained as follows: despite the fact that transport coefficients are positive and the density gradient is negative, which push the ions outwards, the electrostatic potential presents a strong gradient at those positions that overcomes the density gradient driven flux and pushes particles inwards. This strong electric field is created in the experiment by the balance between the ion and electron fluxes. It is important to note here that electron transport is not considered in this work.

The value of the outermost point of the flux estimated by DKEsG is doubtful. The negative value is too large and might be due to the uncertainties of the plasma profiles at those points. In particular, the ion temperature is considered flat, thus not contributing to the flux. Additionally, why the electric field does not affect the ion flux in the case of ISDEP could be questioned. Indeed it does, but not strongly enough to reduce the flux and to make it negative because the ion orbits are wide enough to average the zone of strong electric field at those outer positions of TJ-II. As a final conclusion, it can be noted that for low densities and strong radial electric field, NC ordering is violated in a device with a high magnetic ripple, such as TJ-II, thus giving inaccurate results for the particle and heat fluxes. In such a case, more accurate calculations can be obtained using a global Monte Carlo code like ISDEP.

C.4.3. Relation between rotational transform scaling and NC transport in stellarators

Given the flexibility of TJ-II, independent scans of rotational transform ($\iota/2\pi$) and volume can be performed. The magnetic ripple rises with the volume since the plasma is closer to the coils in the larger configurations, being more sensitive to the variation of the magnetic field. So it is possible to explore the effect of both quantities

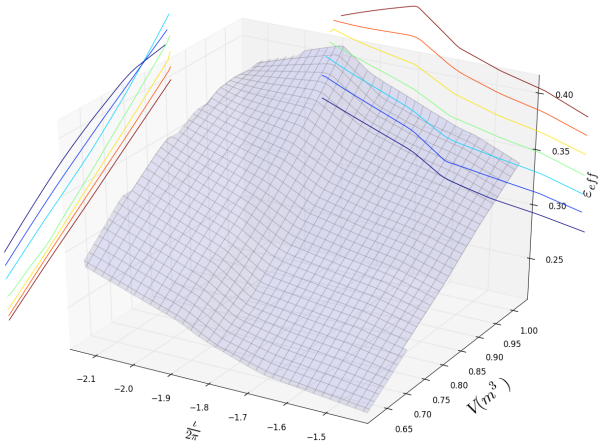


Figure C.8: Representation of the effective ripple (ε_{eff}), as a function of volume and rotational transform in TJ-II for $\rho = 0.650$.

on the effective ripple.

5 different volumes have been taken, with similar values of the rotational transform, and 6 different values of the rotational transform for approximately constant volume. In this way, 30 configurations that scan volume and rotational transform have been considered. The effective ripple ε_{eff} given by Eq. C.14 is calculated for every radial position of these configurations. For this purpose, the DKEsG-Mono outputs obtained from the experiments performed in Section 10.6 are used, with exception of 6 configurations that had previously calculated.

For the evaluation, only 23 radial points are considered for every configuration, and consequently, with slightly different values of $\iota/2\pi$. The effective ripple is shown in Figure C.6 for all the configurations and radii, as a function of volume and $\iota/2\pi$. As the rotational transform is not constant along the minor radius, a curve of ε_{eff} is shown as a function of $\iota/2\pi$ for a given volume. As can be seen, $\iota/2\pi$ is negative in TJ-II since the magnetic field lines twist in the clock-wise direction, but the absolute value of $\iota/2\pi$ must be taken in the above shown scaling laws [302, 303].

To explore the variation of ε_{eff} at different radial positions and, hence, how the NC transport depends on those two parameters, the effective ripple at ρ =0.007, 0.650, 1.000 is plotted. Figure C.7 shows the variation of ε_{eff} as a function of the rotational transform and the volume at the position $\rho = 0$. The effective ripple increases slightly with the volume at this radial position, which is not surprising, given the characteristics of the TJ-II stellarator and the larger excursion of the magnetic axis for larger magnetic configurations. The dependence with the rotational transform is not monotonic and depends on the configuration that is considered.

Figure C.8 shows the same as Figure C.7 for $\rho = 0.650 \approx 2/3$, located in the pressure gradient zone of the plasma, hence having more influence on transport. Again, an increase of the effective ripple with the volume is shown and a less pronounced non-monotonic behaviour with $\iota/2\pi$ appears. In fact, the effective ripple almost rises

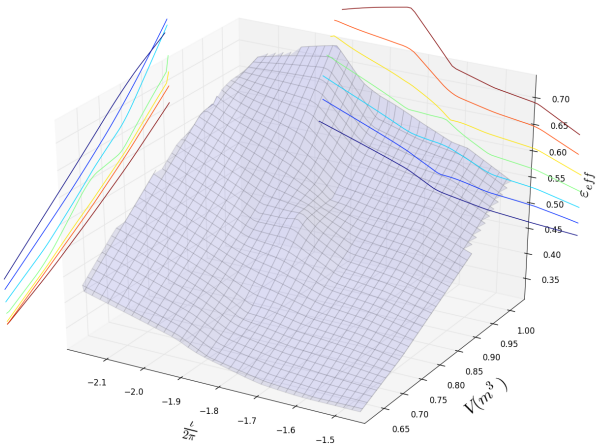


Figure C.9: Representation of the effective ripple (ε_{eff}), as a function of volume and rotational transform in TJ-II for $\rho = 1$.

with the rotational transform, which should provide a dependence with rotational transform opposite to the one observed in the scaling laws and in the semianalytical calculations for tokamaks.

Finally, ε_{eff} is plotted as a function of $\iota/2\pi$ and the volume in Figure C.9 for $\rho = 1$, with the same tendency with volume and a non-monotonic variation with $\iota/2\pi$ for the larger volumes and an increasing ripple for small volumes. The behaviour is opposite to what it is expected form the scaling laws [302, 303] and the semi-analytical estimates for tokamaks [298, 299].

Discussion

The dependence of the confinement with the rotational transform cannot be explained in terms of the NC properties of the device, as calculations for TJ-II show. On the opposite, a global increase of the effective ripple with $\iota/2\pi$ could be extracted for small volumes in TJ-II discharges that goes against the scaling laws that have been obtained up to know. All these facts mean that the NC optimisation helps to the reduction of turbulent transport, but it is not the only ingredient to be considered in the turbulent optimisation of the stellarators. The positive rotational transform dependence of confinement must be attributed to a turbulent mechanism and open new doors to stellarator optimisation.

It has been demonstrated that the NC properties of one device are not necessarily related to the rotational transform profile. In fact, TJ-II characteristics show that the NC transport properties, given by the effective ripple, depend on the volume and not on the rotational transform, since the former is related to the ripple of the magnetic configuration. The point is that the latter parameter is coupled to the poloidal ripple in an axisymmetric tokamak, hence showing influence on the NC transport, while it is decoupled in a stellarator, since the magnetic configuration is created by external

coils and not by the plasma current. Torsatrons present also some relation between the aspect ratio and the rotational transform, which implies some relation between NC optimisation and increasing $\iota/2\pi$.

Therefore, the explanation for the improvement of confinement with the rotational transform cannot be given by NC transport. It is necessary to explore the turbulence properties of the configuration in relation with such a parameter. This assessment implies that the reduction of turbulent transport in a stellarator cannot be performed only by optimising the NC transport, since there are other phenomena that must be considered.

Bibliography

[1] I. Foster, "What Is the Grid? A Three Point Checklist," *GRID-today*, vol. 1, no. 6, 2002. [Online]. Available: http://www.gridtoday.com

[2] I. Foster, Y. Zhao, I. I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop (GCE '08)*. Austin, TX, USA.: IEEE, 12–16 November 2008, pp. 1 – 10. doi : 10.1109/GCE.2008.4738445

[3] M. Garey and D. Johnson, *Computers and Intractibility: A guide to the Theory of NP-completeness*. New York: W. H. Freeman&Co, 1979.

[4] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *10th IEEE International Symposium on High Performance Distributed Computing*, San Francisco, USA, 07–09 August 2001, pp. 181–194. doi : 10.1109/HPDC.2001.945188

[5] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J. P. Navarro, "GLUE Specification v. 2.0," March 2009, GFD 147. [Online]. Available: http://www.ogf.org/documents/GFD.147.pdf

[6] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley- Interscience, April 1991.

[7] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor: A Hunter of Idle Workstations," in *8th International Conference on Distributed Computing Systems*, San José, California, June 1988, pp. 104–111. doi : 10. 1109/DCS. 1988. 12507

[8] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G : A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, July 2002. doi : 10.1023/A:1015617019423

[9] P. Andreetto, S. Andreozzi, G. Avellino, S. Beco, A. Cavallini, M. Cecchi, V. Ciaschini, A. Dorise, F. Giacomini, A. Gianelle, U. Grandinetti, A. Guarise, A. Krop, R. Lops, A. Maraschini, V. Martelli, M. Marzolla, M. Mezzadri, E. Molinari, S. Monforte *et al.*, "The gLite workload management system," *Journal of Physics : Conference Series*, vol. 119, no. 6, p. 062007, 2008. doi : 10.1088/1742-6596/119/6/062007

[10] E. Huedo, R. S. Montero, and I. M. Llorente, "A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 252–261, February 2007. doi : 10.1016/j.future.2006.07.013

[11] J. Montes, A. Sánchez, J. J. Valdés, M. S. Pérez, and P. Herrero, "Finding order in chaos: a behavior model of the whole grid," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 11, August 2010. doi : 10.1002/cpe.1490

[12] T. Glatard and S. Camarasu-Pop, "A model of pilot-job resource provisioning on production grids," *Parallel Computing*, vol. 37, no. 10–11, pp. 684–692, October–November 2011. doi : 10.1016/j.parco.2011.04.001

[13] R. M. Piro, A. Guarise, G. Patania, and A. Werbrouck, "Using historical accounting information to predict the resource usage of grid jobs," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 499–510, May 2009. doi : 10.1016/j.future.2008.11.003

[14] Z. Yu, "Toward Practical multi-workflow Scheduling in Cluster and Grid Environments," Ph.D. dissertation, Wayne State University, Detroit, USA, 2009.

[15] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, Jaunary 2006. doi : 10.1007/s10723-005-9010-8

[16] D. Spiga, "CMS workload management," *Nuclear Physics B - Proceedings Supplements*, vol. 172, pp. 141–144, October 2007. doi : 10.1016/j.nuclphysbps.2007.08.061

[17] A. Tsaregorodtsev, M. Bargiotti, N. Brook, A. C. Ramo, G. Castellani, P. Charpentier, C. Cioffi, J. Closier, R. G. Diaz, G. Kuznetsov, Y. Y. Li, R. Nandakumar, S. Paterson, R. Santinelli, A. C. Smith, M. S. Miguelez, and S. G. Jimenez, "DIRAC: A Community Grid Solution," *Journal Physics : Conference Series*, vol. 119, no. 6, p. 062048, 2008. doi : 10.1088/1742-6596/119/6/062048

[18] P. Saiz, L. Aphecetche, P. Bunčić, R. Piskać, J. E. Revsbech, and V. Šego, "AliEn–ALICE environment on the GRID," *Nuclear Instruments and Methods in Physics Research A*, vol. 502, pp. 437–440, 2003. doi : 10.1016/S0168-9002(03)00462-5

[19] P. Nilsson, J. Caballero, K. De, T. Maeno, M. Potekhin, and T. Wenaus, "The PanDA System in the ATLAS Experiment," in *XII Advanced Computing and Analysis Techniques in Physics Research (ACAT'08)*. Erice, Italy: SISSA PoS, Nov. 2008, pp. 27–1–27–8.

[20] J. Díaz, S. Reyes, A. Niño, and C. Muñoz-Caro, "Derivation of self-scheduling algorithms for heterogeneous distributed computer systems: Application to internet-based grids of computers," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 617–626, June 2009. doi : 10.1016/j.future.2008.12.003

[21] F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future Generation Computer Systems*, vol. 26, no. 4, pp. 608–621, April 2010. doi : 10.1016/j.future.2009.11.005

[22] Y. Gao, H. Rong, and J. Z. Huang, "Adaptive grid job scheduling with genetic algorithms," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 151–161, January 1 2005. doi : 10.1016/j.future.2004.09.033

[23] J. T. Mościcki, "Distributed analysis environment for HEP and interdisciplinary applications," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 502, no. 2–3, pp. 426–429, 2003. doi : 10.1016/S0168-9002(03)00459-5

[24] I. Sfiligoi, "glideinWMS - A generic pilot-based Workload Management System," *Journal of Physics: Conference Series*, vol. 119, p. 062044, 2008. doi : 10. 1088/1742-6596/119/6/062044

[25] P. Buncic, J. F. Grosse-Oetringhaus, A. Peters, and P. Saiz, "The Architecture of the AliEn System," in *Computing in High Energy and Nuclear Physics conference (CHEP'04)*. Interlaken, Switzerland: A. Aimar et al. CERN, Geneva 2005. CERN-2005-02, 27th September - 1st October 2004, pp. 951–954.

[26] A. Luckow, L. Lacinski, and S. Jha, "Saga big job: an extensible and interoperable pilot-job abstraction for distributed applications and systems," in *10th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*. Melbourne, Australia: IEEE Computer Society, 17–20 May 2010, pp. 135–144. doi : 10.1109/CCGRID.2010.91

[27] E. Urbah, P. Kacsuk, Z. Farkas, G. Fedak, G. Kecskemeti, O. Lodygensky, and et al., "EDGeS: Bridging EGEE to BOINC and XtremWeb," *Journal of Grid Computing*, vol. 7, no. 3, pp. 335–354, September 2009. doi : 10.1007/s10723-009-9137-0

[28] M. Silberstein, A. Sharov, D. Geiger, and A. Schuster, "GridBot: execution of bags of tasks in multiple grids," in *Conference on High Performance Computing Networking, Storage and Analysis (SC'09)*. Portland, Oregon, USA: ACM New York, Nov. 2009, pp. 1–12. doi : 10.1145/1654059.1654071

[29] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. M. Badia, "ServiceSs: An Interoperable Programming Framework for the Cloud," *J. Grid Computing*, vol. 12, no. 1, pp. 67–91, March 2014. doi : 10.1007/s10723-013-9272-5

[30] A. Lorca, J. Martín-Caro, R. Núnéz-Ramírez, and J. Martínez-Salazar, "Merging on-demand HPC resources from Amazon EC2 with the grid: a case study of a Xmipp application," *Computing and Informatics*, vol. 31, no. 1, pp. 17–30, 2012.

[31] J. Kovács, A. C. Marosi, A. Visegrádi, Z. Farkas, P. Kacsuk, and R. Lovas, "Boosting gLite with cloud augmented volunteer computing," *Future Generation Computer Systems*, vol. 43–44, pp. 12–23, 2015. doi : 10.1016/j.future.2014.10.005

[32] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 4th ed. Springer, 2012.

[33] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS Conference Proceedings. Spring Joint Computer Conference*, vol. 30, Atlantic City, USA, 18–20 April 1967, pp. 483–485. doi : 10.1145/1465482.1465560

[34] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 0123858801, 9780123858801

[35] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, June 2009. doi : 10.1016/j.future.2008.12.001

[36] R. Bolze and E. Deelman, *Exploiting the Cloud of Computing Environments: an application's Perspective*, in *Cloud Computing and Software Services. Theory and Techniques*. CRC Press. Taylor & Francis Group, 2011, ch. 8. ISBN 978-1-4398-0316-5

[37] A. Shawish and M. Salama, "Cloud computing: Paradigms and technologies," in *Intercooperative Collective Intelligence: Techniques and Applications*, ser. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2014, vol. 495, pp. 39–67. doi : 10.1007/978-3-642-35016-0_2

[38] E. Christoforou, A. F. Anta, C. Georgiou, M. A. Mosteiro, and A. Sánchez, "Applying the dynamics of evolution to achieve reliability in master–worker computing," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 17, pp. 2363–2380, December 2013. doi : 10.1002/cpe.3104

[39] J. Vazquez-Poletti, R. Moreno-Vozmediano, R. Montero, E. Huedo, and I. Llorente, "Solidifying the foundations of the cloud for the next generation software engineering," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2321 – 2326, September 2013. doi : 10.1016/j.jss.2013.05.063

[40] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, January 2008. doi : 10.1145/1327452.1327492

[41] K. Laskey, P. Brown, J. A. Estefan, F. G. McCabe, and D. Thornton, "Reference Architecture Foundation for Service Oriented Architecture 1.0," 04 December 2012, soa-ra-v1.0-cs0. [Online]. Available: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.html

[42] M. Rodríguez, D. Tapiador, J. Fontan, E. Huedo, R. Montero, and I. Llorente, "Dynamic provisioning of virtual clusters for grid computing," in *3rd Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC08)*, ser. Euro-Par 2008 Workshops - Parallel Processing. Lecture Notes in Computer Science, vol. 5415, 2009, pp. 23–32. doi : 10.1007/978-3-642-00955-6_4

[43] C. Vázquez, E. Huedo, R. S. Montero, and I. M. Llorente, "On the use of clouds for Grid resource provisioning," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 600–605, 2011. doi : 10.1016/j.future.2010.10.003

[44] S. Sehgal, M. Erdelyi, A. Merzky, and S. Jha, "Understanding application-level interoperability: Scaling-out MapReduce over high-performance grids and clouds," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 590–599, May 2011. doi : 10.1016/j.future.2010.11.001

[45] J. Conejero, O. Rana, P. Burnap, J. Morgan, B. Caminero, and C. Carrión, "Analyzing Hadoop power consumption and impact on application QoS," *Future Generation Computer Systems*, p. (Available Online), 2015. doi : 10.1016/j.future.2015.03.009

[46] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, no. 6, pp. 37–46, June 2002. doi : 10.1109/MC.2002.1009167

[47] M. Riedel, E. Laure, T. Soddemann, L. Field, and et al., "Interoperation of world-wide production e-Science infrastructures," *Concurrency Computat.: Pract. Exper.*, vol. 21, no. 8, pp. 961–990, June 2009. doi : 10.1002/cpe.1402

[48] A. Kertesz, *Characterizing Cloud Federation Approaches*, in *Cloud Computing (Challenges, Limitations and R&D Solutions)*, ser. Computer Communications and Networks. Springer, Oct. 2014, ch. 12, pp. 277–296. doi : 10.1007/978-3-319-10530-7_12

[49] N. Grozev and R. Buyya, "Inter-Cloud architectures and application brokering: taxonomy and survey," *Softw: Pract. Exper.*, vol. 44, pp. 369–390, 2014. doi : 10.1002/spe.2168

[50] I. Foster and C. Kesselman, "Globus: a Metacomputing Infrastructure Toolkit," *International Journal of High Performance Computing Applications*, vol. 11, no. 2, pp. 115–128, June 1997. doi : 10.1177/109434209701100205

[51] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," in *Job Scheduling Strategies for Parallel Processing(IPPS/SPDP '98)*, ser. Lecture Notes in Computer Science, vol. 1459. Orlando, USA: Springer, 1998, pp. 62–82. doi : 10.1007/BFb0053981

[52] W. Allcock, J. Bresnahan, R. K. M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus Striped GridFTP Framework and Server," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC'05)*. Washington, USA: IEEE CS Press, 12–18 November 2005, pp. 54–64. doi : 10.1109/SC.2005.72

[53] B. Jones, "An Overview of the EGEE Project," in *Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures. 6th Thematic Workshop of the EU Network of Excellence DELOS, June 24-25, 2004.*, ser. Lecture Notes in Computer Science, Cagliari, Italy, 2005, vol. 3664, pp. 1–8. doi : 10.1007/11549819_1

[54] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen, "Advanced Resource Connector middleware for lightweight computational Grids," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 219–240, February 2007. doi : 10.1016/j.future.2006.05.008

[55] D. W. Erwin, "UNICORE-a Grid computing environment," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1395–1410, November 2002. doi : 10.1002/cpe.691

[56] C. Aiftimiei, P. Andreetto, S. Bertocco, S. Fina, A. Dorigo, E. Frizziero, and et al., "Design and implementation of the gLite CREAM jobmanagement service," *Future Generation Computer Systems*, vol. 26, no. 4, pp. 654–667, Apr. 2010. doi : 10.1016/j.future.2009.12.006

[57] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer, "OGSA basic execution service version 1.0," Global Grid Forum, Tech. Rep., 2007, GFD 108. [Online]. Available: http://www.ogf.org/documents/GFD.108.pdf

[58] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific Cloud Computing: Early Definition and Experience," in *10th IEEE International Conference on High Performance Computing and Communications (HPCC'08)*, Dalian, China, 25-27 September 2008, pp. 825–830. doi : 10.1109/HPCC.2008.38

[59] A. Edmonds, T. Metsch, A. Papaspyrou, and A. Richardson, "Toward an open cloud standard," *IEEE Internet Computing*, vol. 16, no. 4, pp. 15–25, 2012. doi : 10.1109/MIC.2012.65

[60] G. S. P. Kacsuk, "Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal," *Journal of Grid Computing*, vol. 3, no. 3–4, pp. 221–238, September 2005. doi : 10.1007/s10723-005-9012-6

[61] M. A. ao, R. Buyya, and S. Venugopal, "InterGrid: A Case for Internetworking Islands of Grids," *Concurrency and Computation: Practice & Experience*, vol. 20, no. 8, pp. 997–1024, 10 June 2008. doi : 10.1002/cpe.1249

[62] D. Petcu and J. L. V.-P. (Eds.), *European Research Activities in Cloud Computing*. The address of the publisher: Cambridge Scholars Publishing, 2012. ISBN 1-4438-3507-2, 978-1-4438-3507-7

[63] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The Reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 1–11, July 2009. doi : 10.1147/JRD.2009.5429058

[64] A. Simón, E. Freire, R. Rosende, I. Díaz, A. Feijóo, P. Rey, J. López-Cacheiro, and C. Fernández, "EGI FedCloud Task Force," in *6th Grid Iberian Infrastructure Conference (IBERGRID'12)*, Lisbon, Portugal, Nov. 7th-9th 2012, pp. 183–194.

[65] A. J. Rubio-Montero, E. Huedo, F. Castejón, and R. Mayo-García, "GWpilot: Enabling multi-level scheduling in distributed infrastructures with GridWay and pilot jobs," *Future Generation Computer Systems*, vol. 45, pp. 25–52, April 2015. doi : 10.1016/j.future.2014.10.003

[66] A. Gómez-Iglesias, M. A. Vega-Rodríguez, and F. Castejón, "Distributed and asynchronous solver for large CPU intensive problems," *Applied Soft Computing*, vol. 13, pp. 2547–2556, 2013. doi : 10.1016/j.asoc.2012.11.031

[67] A. J. Rubio-Montero, M. A. Rodríguez-Pascual, and R. Mayo-García, "Evaluation of an adaptive framework for resilient Monte Carlo executions," in *30th ACM/SIGAPP Symposium On Applied Computing (SAC'15)*. Salamanca, Spain: ACM New York, 13–17 April 2015, pp. 448–455. doi : 10.1145/2695664.2695890

[68] J. T. Mościcki, "Understanding and Mastering Dynamics in Computing Grids: Processing Moldable Tasks with User-Level Overlay," Ph.D. dissertation, Universiteit van Amsterdam, Nederlands, April 2011.

[69] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*, ser. Monographs of Applied Probability and Statistics. London: Chapman and Hall, 1964. doi : 10.1007/978-94-009-5819-7

[70] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods*. Germany: WILEY-VCH, 2008, 2nd Edition.

[71] D. W. O. Rogers, B. A. Faddegon, G. X. Ding, C.-M. Ma, J. Wei, and T. R. Mackie, "BEAM: A Monte Carlo code to simulate radiotherapy treatment units," *Med. Phys.*, vol. 22, pp. 503–524, 1995. doi : 10.1118/1.597552

[72] J. R. Vélez., "Analysis of the air fluorescence induced by electrons for application to cosmic ray detection," Ph.D. dissertation, Universidad Complutense de Madrid, Madrid, Spain, 2011.

[73] M. Rodríguez-Pascual, A. Bustos, F. Castejón, I. M. Llorente, M. Tereshchenko, and R. Mayo-García, "Simulations of fast ions distribution in stellarators based on coupled monte carlo fuelling and orbit codes," *Plasma Physics and Controlled Fusion*, vol. 55, no. 8, p. 085014, June 2013. doi : 10.1088/0741-3335/55/8/085014

[74] P. Arce, J. I. Lagares, L. Harkness, D. Pérez-Astudillo, M. Cañadas, P. Rato, M. de Prado, Y. Abreu, G. de Lorenzo, M. Kolstein, and A. Díaz, "GAMOS: A framework to do Geant4 simulations in different physics fields with an user-friendly interface," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 735, pp. 304–313, Jaunary 2014. doi : 10.1016/j.nima.2013.09.036

[75] E. Huedo, R. S. Montero, and I. M. Llorente, "Experiences on Adaptive Grid scheduling of Parameter Sweep Applications," in *Proc. 12th Euromicro Conference on Parallel, Distributed and Network-based Processing*. Washington: IEEE CS Press, 2004, pp. 28–33. doi : 10.1109/EMPDP.2004.1271423

[76] M. A. Iverson and F. Özgüner, "Hierarchical, competitive scheduling of multiple DAGs in a dynamic heterogeneous environment," *Distributed Systems Engineering*, vol. 6, no. 3, p. 112, September 1999. doi : 10.1088/0967-1846/6/3/303

[77] K. Jensen, *An introduction to the theoretical aspects of Coloured Petri Nets*, in *A Decade of Concurrency Reflections and Perspectives*, ser. Lecture Notes in Computer Science. Springer, 1994, vol. 803, pp. 230–272. doi : 10.1007/3-540-58043-3_21

[78] R. David and H. Alla, "Petri nets for modeling of dynamic systems: A survey," *Automatica*, vol. 30, no. 2, pp. 175–202, February 1994. doi : 10.1016/0005-1098(94)90024-8

[79] L. Moreau, B. Ludäscher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. C. JR., B. Clifford, S. Cohen, S. Cohen-Boulakia, S. Davidson, E. Deelman, L. Digiampietri, I. Foster, J. Freire, J. Frew, J. Futrelle, T. Gibson, Y. Gil, C. Goble *et al.*, "Special Issue: The First Provenance Challenge," *Concurrency and Computation: Practice & Experience*, vol. 20, no. 5, pp. 409–418, April 2008. doi : 10.1002/cpe.1233

[80] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of Data Grids for distributed data sharing, management, and processing," *ACM Computing Surveys*, vol. 38, no. 1, p. 3, 2006. doi : 10.1145/1132952.1132955

[81] K. Hasham, A.D.Peris, A. Anjum, D. Evans, S. Gowdy, J. Hernandez, E. Huedo, D. Hufnagel, F. van Lingen, R. McClatchey, and S. Metson, "CMS Workflow Execution Using Intelligent Job Scheduling and Data Access Strategies," *IEEE Transactions on Nuclear Science*, vol. 58, no. 3, pp. 1221–1232, June 2011. doi : 10.1109/TNS.2011.2146276

[82] L. Ilijašić and L. Saitta, "Characterization of a Computational Grid As a Complex System," in *Proceedings of the 6th International Conference Industry Session on Grids Meets Autonomic Computing (GMAC '09)*. Barcelona, Spain: ACM New York, 2009, pp. 9–18. doi : 10.1145/1555301.1555303

[83] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, June 2013. doi : 10.1016/j.comnet.2013.04.001

[84] M. Sheikhalishahi, R. Wallace, L. Grandinetti, J. L. Vázquez-Poletti, and F. Guerriero, "A multi-dimensional job scheduling," *Future Generation Computer Systems*, 2015. doi : 10.1016/j.future.2015.03.014 Available Online.

[85] M. Pinedo, *Planning and Scheduling in Manufacturing and Services*, ser. Springer Series in Operations Research. New York: Springer, 2005. doi : 10.1007/b139030

[86] R. B. Cooper, *Introduction to Queueing Theory*, 2nd ed. New York: Elsevier Nord Holland, 1981.

[87] I. Adan and J. Resing, *Queueing Theory.* Eindhoven University of Technology. Department of Mathematics and Computing Science, 2001.

[88] R. Diestel, *Graph Theory*, 4th ed., ser. Graduate Texts in Mathematics. Berlin: Springer-Verlag, 2010, vol. 173. ISBN 978-3-642-14278-9

[89] J. Bang-Jensen and G. Z. Gutin, *Digraphs: Theory, Algorithms and Applications.*, 2nd ed., ser. Springer Monographs in Mathematics. London: Springer-Verlag, 2009. doi : 10.1007/978-1-84800-998-1

[90] A. K. Erlang, "Solution of some problems in the theory of probabilities of signicance in automatic telephone exchanges," *The Post Office Engineer's Journal*, vol. 10, pp. 189–197, 1917.

[91] V. E. Beneš, *General Stochastic Processes in the Theory of Queues.* Massachusetts, USA: Addison Wesley, 1963.

[92] T. G. Robertazzi, "Ten Reasons to Use Divisible Load Theory," *Computer*, vol. 36, no. 5, pp. 63–68, May 2003. doi : 10.1109/MC.2003.1198238

[93] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems," *Cluster Computing*, vol. 6, no. 1, pp. 7–17, January 2003. doi : 10.1023/A:1020958815308

[94] S. Viswanathan, B. Veeravalli, and T. Robertazzi, "Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems," *IEEE Transactions on parallel and Distributed Systems*, vol. 18, no. 10, pp. 1450–1461, October 2007. doi : 10.1109/TPDS.2007.1073

[95] C. Yu and D. C. Marinescu, "Algorithms for Divisible Load Scheduling of Data-intensive Applications," *Journal of Grid Computing*, vol. 8, no. 1, pp. 133–155, March 2010. doi : 10.1007/s10723-009-9129-0

[96] M. Abdullah and M. Othman, "Cost-based Multi-QoS Job Scheduling Using Divisible Load Theory in Cloud Computing," in *International Conference on Computational Science (ICCS 2013)*, ser. Procedia Computer Science, vol. 18. Barcelona, Spain: Elsevier, June 2013, pp. 928–935. doi : 10.1016/j.procs.2013.05.258

[97] M. Moges and T. Robertazzi, "Divisible Load Scheduling and Markov Chain Models," *Computers & Mathematics with Applications*, vol. 52, no. 10–11, pp. 1529–1542, November–December 2006. doi : 10.1016/j.camwa.2006.05.016

[98] Y.-K. Kwoka and I. Ahmadb, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms," *Journal of Parallel and Distributed Computing*, vol. 59, no. 3, pp. 381–422, December 1999. doi : 10.1006/jpdc.1999.1578

[99] W. Sadiq and M. E. Orlowska, "Analyzing process models using graph reduction techniques," *Information Systems*, vol. 25, no. 2, pp. 117–134, April 2000. doi : 10.1016/S0306-4379(00)00012-0

[100] W. M. P. V. der Aalst, "The application of Petri nets to workflow management," *Journal of Circuits, Systems and Computers*, vol. 8, no. 1, p. 21, February 1998. doi : 10.1142/S0218126698000043

[101] J. M. Schopf, *Ten Actions When Grid Scheduling*, in *Grid Resource Management*, ser. International Series in Operations Research & Management Science. Springer US, 2004, vol. 64, ch. 2, pp. 15–23. doi : 10.1007/978-1-4615-0509-9

[102] D. G. Feitelson and A. M. Weil, "Utilization and predictability in scheduling the IBM SP2 with backfilling," in *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*. Orlando, USA: IEEE CS Press, 30 March–3 April 1998, pp. 542 – 546. doi : 10.1109/IPPS.1998.669970

[103] T. Casavant and J. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141–154, Feb 1988. doi : 10.1109/32.4634

[104] F. Dong and S. G. Akl, "Scheduling Algorithms for Grid Computing:State of the Art and Open Problems," School of Computing, Queen's University, Kingston, Ontario, USA, Tech. Rep. 2006-504, January 2006.

[105] J. Yu, R. Buyya, and K. Ramamohanarao, *Workflow Scheduling Algorithms for Grid Computing*, in *Metaheuristics for Scheduling in Distributed Computing Environments*, ser. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2008, vol. 146, ch. 7, pp. 173–214. doi : 10.1007/978-3-540-69277-5_7

[106] I. J. Taylor, E. Deelman, D. Gannon, and M. S. (Eds.), *Workflows for e-Science: Scientific Workflows for Grids*. London: Springer-Verlag, 2007. doi : 10.1007/978-1-84628-757-2

[107] S. Smanchat and K. Viriyapant, "Taxonomies of workflow scheduling problem and techniques in the cloud," *Future Generation Computer Systems*, vol. 52, pp. 1–12, November 2015. doi : 10.1016/j.future.2015.04.019

[108] J. Herrera, "Programming Model for Grid Computing Infrastructures. (in Spanish)," Ph.D. dissertation, Universidad Complutense de Madrid, Madrid, Spain, 2009.

[109] M. Rodríguez-Pascual, R. Mayo-García, and I. M. Llorente, "Montera: a framework for efficient execution of Monte Carlo codes on grid infrastructures," *Computing and Informatics*, vol. 32, no. 1, pp. 113–144, 2013.

[110] R. Montero, E. Huedo, and I. Llorente, "Benchmarking of high throughput computing applications on Grids," *Parallel Computing*, vol. 32, no. 4, pp. 267–279, April 2006. doi : 10.1016/j.parco.2005.12.001

[111] S. Zanikolas and R. Sakellariou, "A taxonomy of grid monitoring systems," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 163–188, January 2005. doi : doi:10.1016/j.future.2004.07.002

[112] A. Ciuffoletti, "A Simple and Generic Interface for a Cloud Monitoring Service," in *CLOSER 2014 Proceedings of the 4th International Conference on Cloud Computing and Services Science*. Barcelona, Spain: SCITEPRESS – Science and Technology Publications, 3 - 5 April 2014, pp. 143–150.

[113] P. Mhashilkar, A. Tiradani, B. Holzman, K. Larson, I. Sfiligoi, and M. Rynge, "Cloud Bursting with GlideinWMS: Means to satisfy ever increasing computing needs for Scientific Workflows," in *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP2013)*, ser. Journal of Physics: Conference Series, vol. 513. IOP Publishing, 2014, p. 032069. doi : 10.1088/1742-6596/513/3/032069

[114] A. Luckow, M. Santcroos, A. Zebrowski, and S. Jha, "Pilot-Data: An abstraction for distributed data," *Journal of Parallel and Distributed Computing*, 2014. doi : 10.1016/j.jpdc.2014.09.009 Available online.

[115] R. Graciani, A. Casajús, A. Carmona, T. Fifield, and M. Sevior, "Belle-DIRAC Setup for Using Amazon Elastic Compute Cloud," *Journal of Grid Computing*, vol. 9, no. 1, pp. 65–79, 2011. doi : 10.1007/s10723-010-9175-7

[116] A. Luckow, M. Santcroos, A. Merzky, O. Weidner, P. Mantha, and S. Jha, "P*: A model of pilot-abstractions," in *8th IEEE International Conference on E-Science (e-Science 2012)*, Chicago, USA, 8–12 October 2012, pp. 1–10. doi : 10.1109/eScience.2012.6404423

[117] J. T. Mościcki, M. Lamannaa, M. Bubak, and P. M. A. Sloot, "Processing moldable tasks on the Grid: Late job binding with lightweight user-level overlay," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 725–736, 2011. doi : 10.1016/j.future.2011.02.002

[118] H. González-Vélez and M. Leyton, "A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers," *Software: Practice and Experience*, vol. 40, no. 12, pp. 1135–1160, November/December 2010. doi : 10.1002/spe.1026

[119] P. Troger, H. Rajic, A. Haas, and P. Domagalski, "Standardization of an API for Distributed Resource Management Systems," in *7th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007)*. Rio de Janeiro, Brazil: IEEE CS Press, 14–17 May 2007, pp. 619–626. doi : 10.1109/CCGRID.2007.109

[120] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. V. Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf., "SAGA: A simple API for Grid Applications. High-Level Application Programming on the Grid," *Computational Methods in Science and Technology*, vol. 12, no. 1, pp. 7–20, 2006.

[121] J. T. Mościcki, F. Brochu, J. Ebke, U. Egede, J. Elmsheuser, K. Harrison, R. Jones, H. Lee, D. Liko, A. Maier, A. Muraru, G. Patrick, K. Pajchel, W. Reece, B. Samset, M. Slater, A. Soroko, C. Tan, D. van der Ster, and M. Williams., "GANGA: A tool for computational-task management and easy access to Grid resources," *Computer Physics Communications*, vol. 180, no. 11, pp. 2303–2316, 2009. doi : 10.1016/j.cpc.2009.06.016

[122] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, G. A, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu, "Web Services Business Process Execution Language Version 2.0," 11 April 2007, wsbpel-v2.0-OS. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[123] S. Krishnan, P. Wagstrom, and G. von Laszewski, "Gsfl: A workflow framework for grid services," Argonne National Laboratory, Argonne, USA, Tech. Rep., 2002.

[124] T. Fahringer, S. Pllana, and A. Villazon, "A-gwl: Abstract grid workflow language," in *Computational Science - ICCS 2004. 4th International Conference, Part II*, ser. Lecture Notes in Computer Science, Kraków, Poland, 6-9 June 2004, vol. 3038, pp. 42–49. doi : 10.1007/978-3-540-24688-6_7

[125] Y. Huang, "JISGA: A Jini-Based Service-Oriented Grid Architecture," *International Journal of High Performance Computing Applications*, vol. 17, no. 3, pp. 317–327, August 2003. doi : 10.1177/1094342003173001

[126] D. Cybok, "A Grid workflow infrastructure," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, pp. 1243–1254, August 2006. doi : 10.1002/cpe.998

[127] W. van der Aalst and A. ter Hofstede, "YAWL: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245–275, June 2005. doi : 10.1016/j.is.2004.02.002

[128] M. Alt, A. Hoheisel, H.-W. Pohl, and S. Gorlatch, "A Grid Workflow Language Using High-Level Petri Nets," in *Parallel Processing and Applied Mathematics, 6th International Conference (PPAM 2005)*, ser. Lecture Notes in Computer Science, vol. 3911. Poznań, Poland: Springer Berlin Heidelberg, 11-14 September,2005 2006, pp. 715–722. doi : 10.1007/11752578_86

[129] Z. Guan, F. Hernández, P. Bangalore, J. Gray, A. Skjellum, V. Velusamy, and Y. Liu, "Grid-Flow: a Grid-enabled scientific workflow system with a Petri-net-based interface," *Concurrency Computat.: Pract. Exper.*, vol. 18, no. 10, pp. 1115–1140, August 2006. doi : 10.1002/cpe.988

[130] J. Qin, T. Fahringer, and S. Pllana, *UML based Grid Workflow Modeling under ASKALON*, in *Distributed and Parallel Systems. From Cluster to Grid Computing.* Springer, 2007, pp. 191–200. doi : 10.1007/978-0-387-69858-8_19

[131] J. K. Nilsen, X. Cai, B. Hoyland, and H. P. Langtangen, "Simplifying the parallelization of scientific codes by a function-centric approach in Python," *Computational Science & Discovery*, vol. 3, no. 1, p. 015003, September 2010. doi : 10.1088/1749-4699/3/1/015003

[132] L. Tomás, A. C. Caminero, O. Rana, C. Carrión, and B. Caminero, "A GridWay-based autonomic network-aware metascheduler," *Future Generation Computer Systems*, vol. 28, no. 7, pp. 1058–1069, July 2012. doi : 10.1016/j.future.2011.08.019

[133] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software: Practice and Experience*, vol. 32, no. 2, pp. 135–164, February 2002. doi : 10.1002/spe.432

[134] A. Kertész and P. Kacsuk, *A Taxonomy of Grid Resource Brokers*, in *Distributed and Parallel Systems.* Springer US, 2007, pp. 201–210. doi : 10.1007/978-0-387-69858-8_20

[135] J. L. Vázquez-Poletti, E. Huedo, R. S. Montero, and I. M. Llorente, "A Comparison Between two Grid Scheduling Philosophies: EGEE WMS and GridWay," *Multiagent and Grid Systems*, vol. 3, no. 4, pp. 429–439, 2007.

[136] D. Lingrand, J. Montagnat, J. Martyniak, and D. Colling, "Optimization of jobs submission on the egee production grid: Modeling faults using workload," *Journal of Grid Computing*, vol. 8, no. 2, pp. 305–321, June 2010. doi : 10.1007/s10723-010-9151-2

[137] E. Huedo, R. S. Montero, and I. M. Llorente, "The GridWay Framework for Adaptive Scheduling and Execution on Grids," *Scalable Computing-Practice and Experience*, vol. 6, pp. 1–8, 2005.

[138] I. Marín, E. Huedo, and I. M. Llorente, "Interoperating Grid Infrastructures with the GridWay Metascheduler," *Concurrency and Computation: Practice & Experience*, vol. 27, no. 9, pp. 2278–2290, June 2015. doi : 10.1002/cpe.2971

[139] C. Vázquez, E. Huedo, R. S. Montero, and I. M. Llorente, "Federation of TeraGrid, EGEE and OSG infrastructures through a metascheduler," *Future Generation Computer Systems*, vol. 26, no. 7, pp. 979–985, July 2010. doi : 10.1016/j.future.2010.04.004

[140] B. B. P. Rao, S. Ramakrishnan, M. R. R. Gopalan, C. Subrata, N. Mangala, and R. Sridharan, "e-Infrastructures in IT: A case study on Indian national grid computing initiative – GARUDA," *Computer Science - Research and Development*, vol. 23, no. 3–4, pp. 283–290, June 2009. doi : 10.1007/s00450-009-0079-3

[141] E. Huedo, R. Montero, and I. Llorente, "Evaluating the reliability of computational grids from the end user's point of view," *Journal of Systems Architecture*, vol. 52, no. 12, pp. 727–736, December 2006. doi : 10.1016/j.sysarc.2006.04.003

[142] P. Tröger and A. Merzky, "Towards Standardized Job Submission and Control in Infrastructure Clouds," *J. Grid Computing*, vol. 12, pp. 111–125, 2014. doi : 10.1007/s10723-013-9275-2

[143] G. F. Anastasi, E. Carlini, M. Coppola, and P. Dazzi, "BROKAGE: A Genetic Approach for QoS Cloud Brokering," in *7th IEEE International Conference on Cloud Computing (IEEE CLOUD 2014)*, Alaska. USA, June 27-July 2 2014, pp. 304–311. doi : 10.1109/CLOUD.2014.49

[144] S. Yangui, I.-J. Marshall, J.-P. Laisne, and S. Tata, "CompatibleOne: The Open Source Cloud Broker," *J. Grid Computing*, vol. 12, no. 1, pp. 93–109, March 2014. doi : 10.1007/s10723-013-9285-0

[145] G. Juve and E. Deelman, "Automating Application Deployment in Infrastructure Clouds," in *Third International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov. 29 2011-Dec 1 2011, pp. 658–665. doi : 10.1109/CloudCom.2011.102

[146] E. Walker, J. P. Gardner, V. Litvin, and E. L. Turner, "Personal adaptive clusters as containers for scientific jobs," *Cluster Computing*, vol. 10, no. 3, pp. 339–350, 2007. doi : 10.1007/s10586-007-0028-5

[147] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Multi-Cloud Deployment of Computing Clusters for Loosely-Coupled MTC Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 924–930, June 2011. doi : 10.1109/TPDS.2010.186

[148] J. Buck, S. Ha, A. E. Lee, and A. E. . D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *International Journal of Computer Simulation*, vol. 4, pp. 155–182, August 1994.

[149] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, May 2009. doi : 10.1016/j.future.2008.06.012

[150] G.Terstyanszky, T. Kukla, T. Kiss, P. Kacsuk, A. Balasko, and Z. Farkas, "Enabling scientific workflow sharing through coarse-grained interoperability," *Future Generation Computer Systems*, vol. 37, pp. 46–59, 2014. doi : 10.1016/j.future.2014.02.016

[151] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, pp. 1039–1065, August 2006. doi : 10.1002/cpe.994

[152] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, A. Laity, J. Jacob, and D. Katz, "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005. doi : 10.1155/2005/128026

[153] T. Oinn, M. Greenwood, M. Addis, M. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, pp. 1067–1100, August 2006. doi : 10.1002/cpe.993

[154] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang, "Programming scientific and distributed workflow with Triana services," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, pp. 1021–1037, August 2006. doi : 10.1002/cpe.992

[155] C. Jin and R. Buyya, "A dataflow system with a local optima-based scheduling for enterprise grids," Grid Computing and Distributed Systems Laboratory, University of Melbourne, Melbourne, Australia, Tech. Rep., November 2007.

[156] P. Kacsuk and G. Sipos, "Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal," *Journal of Grid Computing*, vol. 3, no. 3–4, pp. 221–238, September 2006. doi : 10.1007/s10723-005-9012-6

[157] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, and H.-L. Truong, "ASKALON: A Tool Set for Cluster and Grid Computing," *Concurrency and Computation: Practice & Experience*, vol. 17, no. 2-4, pp. 143–169, February – April 2005. doi : 10.1002/cpe.929

[158] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR," *International Journal of High Performance Computing Applications*, vol. 22, no. 3, pp. 347–360, August 2008. doi : 10.1177/1094342008096067

[159] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, S. Mei-Hui, and K. Vahi, "Characterization of scientific workflows," in *Third Workshop on Workflows in Support of Large-Scale Science (WORKS 2008)*. Austin, USA: IEEE CS Press, 17-17 November 2008, pp. 1–10. doi : 10.1109/WORKS.2008.4723958

[160] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, vol. 1, no. 1, pp. 25–39, March 2003. doi : 10.1023/A:1024000426962

[161] J. L. Vázquez-Poletti, E. Huedo, R. S. Montero, and I. M. Llorente, "Workflow Management in a Protein Clustering Application," in *Proc. 5th Intl. Workshop on Biomedical Computations on the Grid (BioGrid'07) on the 7th IEEE Intl. Symp. on Cluster Computing and the Grid (CCGrid 2007)*, Rio de Janeiro, 14–17 May 2007, pp. 679–684. doi : 10.1109/CCGRID.2007.122

[162] M. Wieczorek, R. Prodan, and T. Fahringer, "Comparison of Workflow Scheduling Strategies on the Grid," in *Parallel Processing and Applied Mathematics.6th International Conference, (PPAM 2005)*, ser. Lecture Notes in Computer Science, Poznań, Poland, 11-14 September, 2005 2006, vol. 3911, pp. 792–800. doi : 10.1007/11752578_95

[163] M. A. S. M. M. López, E. Heymann, "Sensitivity Analysis of Workflow Scheduling on Grid Systems," *Scalable Computing: Practice and Experience*, vol. 8, no. 3, pp. 301–311, 2007.

[164] Y. C. Lee, H. Han, A. Y. Zomaya, and M. Yousif, "Resource-efficient workflow scheduling in clouds," *Knowledge-Based Systems*, vol. 80, pp. 153–162, May 2015. doi : 10.1016/j.knosys.2015.02.012

[165] R. Tolosana-Calasanz, J. A. Bañares, C. Pham, and O. F. Rana, "Enforcing QoS in scientific workflow systems enacted over Cloud infrastructures," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1300–1315, September 2012. doi : 10.1016/j.jcss.2011.12.015

[166] E. N. Alkhanak, S. P. Lee, and S. U. R. Khan, "Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities," *Future Generation Computer Systems*, vol. 50, pp. 3–21, September 2015. doi : 10.1016/j.future.2015.01.007

[167] C.-C. Hsu, K.-C. Huang, and F.-J. Wang, "Online scheduling of workflow applications in grid environments," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 860–870, June 2011. doi : 10.1016/j.future.2010.10.015

[168] M. Gendreau and J.-Y. P. (Eds.), *Handbook of Metaheuristics*, 2nd ed., ser. International Series in Operations Research & Management Science. Springer US, 2010, vol. 146. doi : 10.1007/978-1-4419-1665-5

[169] A. B. Downey, "A model for speedup of parallel programs," Computer Science Division (EECS), University of California, Berkeley, USA, Tech. Rep., January 1997.

[170] S. Jang, X. Wu, V. Taylor, G. Mehta, K. Vahi, and E. Deelman, "Using performance prediction to allocate grid resources," Texas A&M University, GriPhyN, College Station, Texas, USA., Tech. Rep., 2004.

[171] Y. Li and M. Mascagni, "Grid-based monte carlo application," in *Grid Computing – GRID 2002*, ser. Lecture Notes in Computer Science, Baltimore, USA, November 2002, vol. 2536, pp. 13–24. doi : 10.1007/3-540-36133-2_2

[172] D. . P. da Silva, W. Cirne, and F. V. Brasileiro, "Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids," in *Euro-Par 2003 Parallel Processing. 9th International Euro-Par Conference*, ser. Lecture Notes in Computer Science, Klagenfurt, Austria, 26-29 August 2003, vol. 2790, pp. 169–180. doi : 10.1007/978-3-540-45209-6_26

[173] J. L. Vázquez-Poletti, E. Huedo, R. S. Montero, and I. M. Llorente, "CD-HIT Workflow Execution on Grids Using Replication Heuristics," in *8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID).*, Lyon, France, 19–22 May 2008, pp. 735–740. doi : 10.1109/CCGRID.2008.37

[174] J. Chen and Y. Yang, "Activity Completion Duration Based Checkpoint Selection for Dynamic Verification of Temporal Constraints in Grid Workflow Systems," *International Journal of High Performance Computing Applications*, vol. 22, no. 3, pp. 319–329, August 2008. doi : 10.1177/1094342007086229

[175] R. Buyya and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *Concurrency and Computation: Practice & Experience*, vol. 14, no. 13-15, pp. 1175–1220, November - December 2002. doi : 10.1002/cpe.710

[176] D. Klusáček, L. Matyska, and H. Rudová, "Alea - grid scheduling simulation environment," in *Parallel Processing and Applied Mathematics. 7th International Conference (PPAM 2007)*, ser. Lecture Notes in Computer Science, Gdansk, Poland, 9-12 September, 2007 2008, vol. 4967, pp. 1029–1038. doi : 10.1007/978-3-540-68111-3_109

[177] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "Journal of Grid Computing," *iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator*, vol. 10, no. 1, pp. 185–209, March 2012. doi : 10.1007/s10723-012-9208-5

[178] M. R. J. Novotny and O. Wehrens, "GridSphere: a portal framework for building collaborations," *Concurrency and Computation: Practice & Experience*, vol. 16, no. 5, pp. 503–513, April 2004. doi : 10.1002/cpe.829

[179] D. Szejnfeld, P.Dziubecki, P. Kopta, M. Krysinski, T. Kuczynski, K. Kurowski, B. Ludwiczak, T. Piontek, D. Tarnawczyk, M. Wolniewicz, P. Domagalski, J. Nabrzyski, and K. Witkowski, "Vine ToolkitÑTowards portal based production solutions for scientific and engineering communities with grid-enabled resources support," *Scalable Computing: Practice and Experience*, vol. 11, no. 2, pp. 161–172, 2010.

[180] A. Andronico, R. Barbera, A. Falzone, P. Kunszt, G. L. Re, A. Pulvirenti, and A. Rodolico, "GENIUS: a simple and easy way to access computational and data grids," *Future Generation Computer Systems*, vol. 19, no. 6, pp. 805–813, August 2003. doi : 10.1016/S0167-739X(03)00061-X

[181] V. Ardizzone, R. Barbera, A. Calanducci, M. Fargetta, E. Ingrà, I. Porro, G. L. Rocca, S. Monforte, R. Ricceri, R. Rotondo, D. Scardaci, and A. Schenone, "The DECIDE Science Gateway," *Journal of Grid Computing*, vol. 10, no. 4, pp. 689–707, December 2012. doi : 10.1007/s10723-012-9242-3

[182] S. Shahand, M. Santcroos, A. H. C. van Kampen, and S. D. Olabarriaga, "A Grid-Enabled Gateway for Biomedical Data Analysis," *Journal of Grid Computing*, vol. 10, no. 4, pp. 725–742, December 2012. doi : 10.1007/s10723-012-9233-4

[183] G. Codispoti, C. Mattia, A. Fanfani, F. Fanzago, F. Farina, C. Kavka, S. Lacaprara, V. Miccio, D. Spiga, and E. Vaandering, "CRAB: A CMS application for distributed analysis," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 5, pp. 2850–2858, 2009. doi : 10.1109/TNS.2009.2028076

[184] D. Evans, A. Fanfani, C. Kavka, F. van Lingen, G. Eulisse, W. Bacchi, G. Codispoti, D. Mason, N. D. Filippis, J. M. Hernández, and P. Elmer, "The CMS Monte Carlo Production System: Development and Design," *Nuclear Physics B - Proceedings Supplements*, vol. 177-178, pp. 285–286, 2008. doi : 10.1016/j.nuclphysbps.2008.02.001

[185] A. Fanfani, A. Afaq, J. A. Sanches, J. Andreeva, G. Bagliesi, L. Bauerdick, S. Belforte, P. Bittencourt, K. Bloom, B. Blumenfeld, and et al, "Distributed Analysis in CMS," *Journal of Grid Computing*, vol. 8, no. 2, pp. 159–179, 2010. doi : 10.1007/s10723-010-9152-1

[186] D. Groep, O. Koeroo, and G. Venekamp, "gLExec: gluing Grid computing to the Unix world," *Journal of Physics : Conference Series*, vol. 119, no. 6, p. 062032, 2008. doi : 10.1088/1742-6596/119/6/062032

[187] I. Sfiligoi, "CDF computing," *Computer Physics Communications*, vol. 177, no. 1–2, pp. 235–238, July 2007. doi : 10.1016/j.cpc.2007.02.055

[188] E. Deelman, G. Singh, M. H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005. doi : 10.1155/2005/128026

[189] G. Juve, E. Deelman, K. Vahi, and G. Mehta, "Experiences with resource provisioning for scientific workflows using Corral," *Scientific Programming*, vol. 18, no. 2, pp. 77–92, 2010. doi : 10.3233/SPR-2010-0300

[190] M. Altunay, P. Avery, K. Blackburn, B. Bockelman, M. Ernst, D. Fraser, R. Quick, R. Gardner, S. Goasguen, T. Levshina, and et al., "A Science Driven Production Cyberinfrastructure – the Open Science Grid," *Journal of Grid Computing*, vol. 9, no. 2, pp. 201–218, 2011. doi : 10.1007/s10723-010-9176-6

[191] E. Michon, J. Gossa, S. Genaud, M. Frincu, and A. Burel, "Porting Grid Applications to the Cloud with Schlouder," in *IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, Bristol, United Kingdom, 2–5 Dec. 2013, pp. 505–512. doi : 10.1109/CloudCom.2013.73

[192] K. Torberntsson and Y. Rydin, "A Study of Configuration Management Systems. Solutions for Deployment and Configuration of Software in a Cloud Environment," June 2014, B.S. Thesis. Uppsala University. Sweden.

[193] R. S. Montero, R. Moreno-Vozmediano, and I.M.Llorente, "An elasticity model for High Throughput Computing clusters," *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 750–757, June 2011. doi : 10.1016/j.jpdc.2010.05.005

[194] V. Méndez, A. Casajús, V. Fernández, R. Graciani, and G. Merino, "Rafhyc: an Architecture for Constructing Resilient Services on Federated Hybrid Clouds," *J. Grid Computing*, vol. 11, pp. 753–770, 2013. doi : 10.1007/s10723-013-9279-y

[195] D. Bradley, O. Gutsche, K. Hahn, B. Holzman, S. Padhi, H. Pi, D. Spiga, I. Sfiligoi, E. Vaandering, and F. Würthwein, "Use of glide-ins in CMS for production and analysis," *Journal of Physics : Conference Series*, vol. 219, no. 7, p. 072013, 2010. doi : 10.1088/1742-6596/219/7/072013

[196] M. Cinquilli, A. F. G. Codispoti, F. Fanzago, F. Farina, S. L. J. Hernández, H. Riahi, D. Spiga, E. Vaandering, and S. Wakefield, "Tools to use heterogeneous Grid schedulers and storage system," in *13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research, (ACAT2010)*. Jaipur, India: SISSA PoS, February 22–27 2010, pp. 29–1–29–13.

[197] A. Tsaregorodtsev, V. Garonne, and I. Stokes-Rees, "DIRAC: A scalable lightweight architecture for high throughput computing," in *5th IEEE/ACM International Workshop on Grid Computing (GRID'04)*. Pittsburgh, USA: IEEE CS Press, Nov. 2004, pp. 19–25. doi : 10.1109/GRID.2004.22

[198] V. M. Muñoz, R. V. F. Albor, A. Casajús, T. F. Pena, G. Merino, and J. J. Saborido, "The Integration of CloudStack and OCCI/OpenNebula with DIRAC," *Journal of Physics: Conference Series*, vol. 396, p. 032075, 2012. doi : 10.1088/1742-6596/396/3/032075

[199] P. Buncic, C. Aguado, J. Blomer, A. Harutyunyan, and M. Mudrinic, "A practical approach to virtualization in HEP," *The European Physical Journal Plus*, vol. 126, no. 1, p. 13, January 2011. doi : 10.1140/epjp/i2011-11013-1

[200] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, July 2004. doi : 10.1016/j.parco.2004.04.001

[201] X. Zhao, J. Hover, T. Wlodek, T. Wenaus, J. Frey, T. Tannenbaum, and M. Livny, "PanDA Pilot Submission using Condor-G: Experience and Improvements," *Journal of Physics : Conference Series*, vol. 331, no. 7, p. 072069, 2011. doi : 10.1088/1742-6596/331/7/072069

[202] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, F. Furano, A. Grigoras, C. Grigoras, P. M. Lorenzo, A. J. Peters, and P. Saiz, "The ALICE Workload Management System: Status before the real data taking," *Journal of Physics : Conference Series*, vol. 219, no. 6, p. 062004, 2010. doi : 10.1088/1742-6596/219/6/062004

[203] E. Carona, V. Garonne, and A. Tsaregorodtsev, "Definition, modelling and simulation of a Grid computing scheduling system for high throughput computing," *Future Generation Computer Systems*, vol. 23, no. 8, pp. 968–976, November 2007. doi : 10.1016/j.future.2007.04.008

[204] S. K. Paterson and A. Tsaregorodtsev, "DIRAC Optimized Workload Management," *Journal of Physics : Conference Series*, vol. 119, no. 6, p. 062040, 2008. doi : 10.1088/1742-6596/119/6/062040

[205] P. Nilsson, "Experience from a pilot based system for ATLAS," *Journal of Physics : Conference Series*, vol. 119, no. 6, p. 062038, 2008. doi : 10.1088/1742-6596/119/6/062038

[206] P.-H. Chiu and M. Potekhin, "Pilot factory – a Condor-based system for scalable Pilot Job generation in the Panda WMS framework," *Journal of Physics : Conference Series*, vol. 219, no. 6, p. 062041, 2010. doi : 10.1088/1742-6596/219/6/062041

[207] S. Camarasu-Pop, T. Glatard, R. F. da Silva, P. Gueth, D. Sarrut, and H. Benoit-Cattin, "Monte Carlo simulation on heterogeneous distributed systems: A computing framework with parallel merging and checkpointing strategies," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 728–738, March 2013. doi : 10.1016/j.future.2012.09.003

[208] I. Stokes-Rees, A. Tsaregorodtsev, and V. Garonne, "DIRAC Lightweight Information and Monitoring Services using XML-RPC and Instant Messaging," in *Computing in High Energy and Nuclear Physics conference (CHEP'04)*. Interlaken, Switzerland: A. Aimar et al. CERN, Geneva 2005. CERN-2005-02., 27th September - 1st October 2004, pp. 788–791.

[209] G. Shao, "Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources," Ph.D. dissertation, University of Califorina, San Diego, USA, 2001.

[210] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive Computing on the Grid Using AppLeS," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 369–382, April 2003. doi : 10.1109/TPDS.2003.1195409

[211] J.-P. Goux, S. Kulkarni, M. Yoder, and J. Linderoth., "Master–Worker: An Enabling Framework for Applications on the Computational Grid," *Cluster Computing*, vol. 4, pp. 63–70, 2001. doi : 10.1023/A:1011416310759

[212] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova, "A GridRPC Model and API for End-User Applications," June 29 2007, GFD-R.052. [Online]. Available: http://www.ogf.org/documents/GFD.52.pdf

[213] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka, "Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing," *Journal of Grid Computing*, vol. 1, pp. 41–51, 2003. doi : 10.1023/A:1024083511032

[214] H. Rajic, R. Borbst, W. Chan, F. Fersti, J. Gardiner, A. Haas, B. Nitzberg, D. Templeton, J. Tollefsrud, and P. Tröger, "Distributed Resource Management Application

API Specification 1.0," June 2008, GFD 133. [Online]. Available: http://www.ogf.org/documents/GFD.133.pdf

[215] V. V. Korkhov, V. V. Krzhizhanovskaya, and P. M. A. Sloot, "A Grid-Based Virtual Reactor: Parallel performance and adaptive load balancing," *Journal of Parallel and Distributed Computing*, vol. 68, no. 5, pp. 596–608,, 2008. doi : 10.1016/j.jpdc.2007.08.010

[216] V. V. Korkhov, J. T. Mościcki, and V. V. Krzhizhanovskaya, "Dynamic workload balancing of parallel applications with user-level scheduling on the Grid," *Future Generation Computer Systems*, vol. 25, no. 1, pp. 28–34, Jaunary 2009. doi : 10.1016/j.future.2008.07.001

[217] D. P. Anderson., "BOINC: A System for Public-Resource Computing and Storage," in *5th IEEE/ACM Int. Workshop on Grid Computing (GRID'04)*. Pittsburgh, USA: IEEE CS Press, 8 November 2004, pp. 4–10. doi : 10.1109/GRID.2004.14

[218] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, and O. Lodygensky, "Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with Grid," *Future Generation Computer Systems*, vol. 21, no. 3, pp. 417–437, March 2005. doi : 10.1016/j.future.2004.04.011

[219] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: a Fast and Light-weight tasK executiON framework," in *ACM/IEEE Conference on Supercomputing (SC'07)*. Reno, Nevada, USA: ACM New York, 2007, pp. 1–12. doi : 10.1145/1362622.1362680

[220] S. Camarasu-Pop, T. Glatard, J. T. Mościcki, H. Benoit-Cattin, and D. Sarrut, "Dynamic Partitioning of GATE Monte-Carlo Simulations on EGEE," *Journal of Grid Computing*, vol. 8, no. 2, pp. 241–259, June 2010. doi : 10.1007/s10723-010-9153-0

[221] G. Castellani and R. Santinelli, "Job Prioritization and Fair Share in the LHCb experiment," *Journal of Physics : Conference Series*, vol. 119, no. 7, p. 072009, 2008. doi : 10.1088/1742-6596/119/7/072009

[222] S. Camarasu-Pop, T. Glatard, and H. Benoit-Cattin, "Simulating Application Workflows and Services Deployed on the European Grid Infrastructure," in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Delft, Netherlands, 13–16 May 2013, pp. 18–25. doi : 10.1109/CCGrid.2013.13

[223] S. Madougou, S. Shahand, M. Santcroos, A. B. B. van Schaik and, A. van Kampen, and S. Olabarriaga, "Characterizing workflow-based activity on a production e-infrastructure using provenance data," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 1931–1942, October 2013. doi : 10.1016/j.future.2013.04.019

[224] M. A. Rodríguez-Pascual, J. Guasp, F. Castejón, A. J. Rubio-Montero, I. M. Llorente, and R. Mayo, "A Grid version of the Fusion code FAFNER," in *18th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2010)*. Pisa, Italy: IEEE CS Press, 17–19 February 2010, pp. 449–453. doi : 10.1109/PDP.2010.37

[225] M. Rodríguez-Pascual, F. Castejón, A. J. Rubio-Montero, R. Mayo, and I. M. Llorente, "FAFNER2: A comparison between the Grid and the MPI versions of the code," in *Int. Conf. on High Performance Comput. and Simulation (HPCS 2010)*. Caen, France: IEEE CS Press, 28 June–2 July 2010, pp. 78–84. doi : 10.1109/HPCS.2010.5547146

[226] M. Rodríguez-Pascual, J. Guasp, F. Castejón, A. J. Rubio-Montero, I. M. Llorente, and R. Mayo, "Improvements on the Fusion Code FAFNER2," *IEEE Transactions on Plasma Science*, vol. 38, no. 9, pp. 2102–2110, September 2010. doi : 10.1109/TPS.2010.2057450

[227] M. Rodríguez-Pascual, A. J. Rubio-Montero, R. Mayo, A. Bustos, F. Castejón, and I. Llorente, "More Efficient Executions of Monte Carlo Fusion Codes by Means of Montera: The ISDEP Use Case," in *19th Euromicro Int. Conf. on Parallel, Distributed*

and Network-Based Processing (PDP 2011). Ayia Napa, Cyprus: IEEE CS Press, 9–11 February 2011, pp. 380–384. doi : 10.1109/PDP.2011.46

[228] M. Rodíguez-Pascual, D. P. de Lara, E. M. González, A. Gómez, A. J. Rubio-Montero, R. Mayo, and J. Vicent, "Grid computing simulation of superconducting vortex lattice in superconducting magnetic nanostructures," in *Proceedings of the 4th Iberian Grid Infrastructure Conference*, vol. 4. Braga, Portugal: NETBIBLO S.L. (Sta. Cristina, La Coruña, Spain), 24–27 May 2010, pp. 97–109. ISBN 978-84-9745-549-7

[229] M. Rodríguez-Pascual, A. Gómez, R. Mayo-García, D. P. de Lara, E. M. González, A. J. Rubio-Montero, and J. L. Vicent, "Superconducting Vortex Lattice Configurations on Periodic Potentials: Simulation and Experiment," *Superconductivity and Novell Magnetism*, vol. 25, no. 7, pp. 2127–2130, October 2012. doi : 10.1007/s10948-012-1636-8

[230] A. J. Rubio-Montero, L. Flores, F. Castejón, E. Montes, M. Rodríguez-Pascual, and R. Mayo, "Executions of a Drift Kinetic Ecuation solver on Grid," in *18th Euromicro Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2010)*. Pisa, Italy: IEEE CS Press, 17–19 February 2010, pp. 454–459. doi : 10.1109/PDP.2010.40

[231] M. Rodríguez-Pascual, A. J. Rubio-Montero, R. Mayo-García, C. Kanellopoulos, O. Prnjat, D. Darriba, and D. Posada, "A fault tolerant workflow for reproducible research," in *Annual Global Online Conference on Information and Computer Technology (GOCICT 2014)*. Louisville, Kentucky, USA: IEEE CS Press, 3–5 December 2014, pp. 70–75. doi : 10.1109/GOCICT.2014.10

[232] A. J. Rubio-Montero, M. A. Rodríguez-Pascual, and R. Mayo-García, "A simple model to exploit reliable algorithms in cloud federations," *Soft Computing*, p. (Under Review), 2016.

[233] A. J. Rubio-Montero, R. S. Montero, E. Huedo, and I. M. Llorente, "Management of Virtual Machines on Globus Grids using GridWay," in *21st IEEE Int. Parallel and Distributed Processing Symposium (IPDPS 2007)*. Long Beach, USA: IEEE CS Press, 27–30 March 2007, pp. 1–7. doi : 10.1109/IPDPS.2007.370548

[234] A. J. Rubio-Montero, F. Castejón, M. A. Rodríguez-Pascual, E. Montes, and R. Mayo, "Drift Kinetic Equation Solver for Grid (DKEsG)," *IEEE Transactions on Plasma Science*, vol. 38, no. 9, pp. 2093–2101, September 2010. doi : 10.1109/TPS.2010.2055164

[235] R. Isea, J. Chaves, E. Montes, A. J. Rubio-Montero, and R. Mayo, "The evolution of HPV by means of a phylogenetic study," in *Healthgrid Research, Innovation and Business Case. Proceedings of HealthGrid 2009*, ser. Studies in Health Technology and Informatics, vol. 147. Berlin, Germany: IOS Press, 29 June–1 July 2009, pp. 245–250. doi : 10.3233/978-1-60750-027-8-245

[236] R. Isea, E. Montes, A. J. Rubio-Montero, and R. Mayo, "Computational Challenges on Grid Computing for Workflows Applied to Phylogeny," in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living. 10th Int. Work-Conference on Artificial Neural Networks (IWANN 2009)*, ser. Lecture Notes in Computer Science, vol. 5518. Salamanca, Spain: Springer-Verlag, 10–12 June 2009, pp. 1130–1138. doi : 10.1007/978-3-642-02481-8_171

[237] R. Isea, E. Montes, A. J. Rubio-Montero, J. D. Rosales, M. A. Rodríguez-Pascual, and R. Mayo, "Characterization of antigenetic serotypes from the dengue virus in Venezuela by means of Grid Computing," in *Healthgrid Applications and core Technologies. Proceedings of HealthGrid 2010*, ser. Studies in Health Technology and Informatics, vol. 159. Paris, France: IOS Press, 28–30 June 2010, pp. 234–238. doi : 10.3233/978-1-60750-583-9-234

[238] R. Isea, E. Montes, A. J. Rubio-Montero, and R. Mayo, *State-of-Art with PhyloGrid: Grid Computing Phylogenetic Studies on the EELA-2 Project Infrastructure*, in *Grid Computing: Towards a Global Interconnected Infrastructure*, ser. Computer Communications and Networks. Springer London / Heidelberg New York, 2011, pp. 277–291. doi : 10.1007/978-0-85729-676-4_11

[239] A. J. Rubio-Montero, P. Arce, J. I. Lagares, Y. P. Ivanov, D. A. Burbano, G. Díaz, and R. Mayo, "Performance Tests of GAMOS Software on EELA-2 Infrastructure," in *Proceedings of the Second EELA-2 Conference*. Choroní, Venezuela: Editorial CIEMAT (Madrid, Spain), 25–27 November 2009, pp. 379–385. ISBN 978-84-7834-627-1

[240] A. J. Rubio-Montero, F. Castejón, E. Huedo, M. Rodríguez-Pascual, and R. Mayo-García, "Performance improvements for the neoclassical transport calculation on Grid by means of pilot jobs," in *Int. Conf. on High Performance Comput. and Simulation (HPCS 2012)*. Madrid, Spain: IEEE CS Press, 2–6 July 2012, pp. 609–615. doi : 10.1109/HPCSim.2012.6266981

[241] A. J. Rubio-Montero, F. Castejón, E. Huedo, and R. Mayo-García, "A novel pilot job approach for improving the execution of distributed codes: application to the study of ordering in collisional transport in fusion plasmas," *Concurrency and Computation: Practice & Experience*, vol. 27, no. 13, pp. 3220–3244, September 2015. doi : 10.1002/cpe.3301

[242] A. J. Rubio-Montero, E. Huedo, and R. Mayo-García, "Scheduling multiple virtual environments in cloud federations for distributed calculations," *Future Generation Computer Systems*, p. (Under Review), 2016.

[243] A. Lorca, E. Huedo, and I. Llorente, "The Grid[Way] Job Template Manager, a tool for parameter sweeping," *Computer Physics Communications*, vol. 182, no. 4, pp. 1047–1060, April 2011. doi : 10.1016/j.cpc.2010.12.041

[244] W. I. van Rij and S. P. Hirshman, "Variational bounds for transport coefficients in three-dimensional toroidal plasmas," *Phys. Fluids B*, vol. 1, no. 3, pp. 563–569, March 1989. doi : 10.1063/1.859116

[245] D. A. Spong, "Generation and damping of neoclassical plasma flows in stellarators," *Physics of Plasmas*, vol. 12, no. 5, pp. 056–114, May 2005. doi : 10.1063/1.1887172

[246] J. H. Wilkinson and C. Reinsch, *Linear Algebra II*, in *Handbook for Automatic Computation*. Berlin: Springer-Verlag, 1993.

[247] Z. Chen, J. Dongarra, P. Luszczek, and K. Roche, "Self adapting software for numerical linear algebra and LAPACK for clusters," *Parallel Computing*, vol. 29, pp. 1723–1743, 2003. doi : 10.1016/j.parco.2003.05.014

[248] D. Spong, S. Hirshman, J. Lyon, L. Berry, and D. Strickler, "Recent advances in quasi-poloidal stellarator physics issues," *Nuclear Fusion*, vol. 45, no. 8, pp. 918–925, Aug. 2005. doi : 10.1088/0029-5515/45/8/020

[249] A. Gómez-Iglesias, M. A. Vega-Rodríguez, F. Castejón, E. Morales-Ramos, M. Cárdenas-Montes, and J. M. Reynolds, "Grid-based metaheuristics to improve a nuclear fusion device," *Concurrency Computat.: Pract. Exper.*, vol. 22, no. 11, pp. 1476–1493, Aug. 2010. doi : 10.1002/cpe.1497

[250] A. Cappa, D. López-Bruna, F. Castejón, M. Ochando, J. Vázquez-Poletti, and et al., "Dynamic Simulation of the Electron Bernstein Wave Heating under NBI Conditions in TJ-II Plasmas," *Contributions to Plasma Physics*, vol. 51, no. 1, pp. 83–91, January 2011. doi : 10.1002/ctpp.200900060

[251] G. Pereverzev and P. Yushmanov, "ASTRA: Automated System for Transport Analysis," IPP 5/98, Garching, Garching, Germany, Tech. Rep., February 2002.

[252] A. J. Rubio-Montero, E. Huedo, and R. Mayo-García, "User-Guided Provisioning in Federated Clouds for Distributed Calculations," in *Adaptive Resource Management and Scheduling for Cloud Computing (ARMS-CC 2015)*, ser. Lecture Notes in Computer Science, vol. 9438. San Sebastián, Spain: Springer, 20th July 2015, pp. 60–77. doi : 10.1007/978-3-319-28448-4_5

[253] B. Parák, Z. Šustr, F. Feldhaus, P. Kasprzakc, and M. Srbac, "The rOCCI Project: Providing Cloud Interoperability with OCCI 1.1," in *International Symposium on Grids and Clouds (ISGC)*, ser. SISA PoS, Taipei, Taiwan, 23–28 March 2014, pp. 1–15.

[254] H. J. Curnow and B. A. Wichmann, "A synthetic benchmark," *The Computer Journal*, vol. 19, no. 1, pp. 43–49, 1976. doi : 10.1093/comjnl/19.1.43

[255] T. H. Tzen and L. M. Ni, "Trapezoid self-scheduling: a practical scheduling scheme for parallel compilers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, January 1993. doi : 10.1109/71.205655

[256] E. Huedo, R. S. Montero, and I. M. Llorente, "Grid Architecture from a Metascheduling Perspective," *Computer*, vol. 43, no. 7, pp. 51–56, July 2010. doi : 10.1109/MC.2010.91

[257] A. S. McGough, W. Lee, and S. Das, "A standards based approach to enabling legacy applications on the Grid," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 731–743, July 2008. doi : 10.1016/j.future.2008.02.004

[258] C. Germain-Renaud, C. Loomis, J. T. Mościcki, and R. Texier, "Scheduling for Responsive Grids," *Journal of Grid Computing*, vol. 6, no. 1, pp. 15–27, 2008. doi : 10.1007/s10723-007-9086-4

[259] R. Raman, M. Solomon, M. Livny, and A. Roy, "The classads language," in *Grid Resource Management. State of the Art and Future Trends*, ser. International Series in Operations Research & Management Science. Springer US, 2004, vol. 64, ch. 17, pp. 255–270. doi : 10.1007/978-1-4615-0509-9_17

[260] C. Alejaldre, J. J. Alonso, J. Botija, F. Castejón, J. R. Cepero, and et al., "TJ-II project: A flexible Heliac stellarator," *Fusion Technology*, vol. 17, no. 1, pp. 131–139, 1990.

[261] D. Kelsey, "Policy on Grid Multi-User Pilot Jobs. EGI-SPG-PilotJobs-V1_0," 2010, EGI Document 84-v6. [Online]. Available: https://documents.egi.eu/document/84

[262] J. T. Mościcki, M. Woś, M. Lamanna, P. de Forcrand, and O. Philipsen, "Lattice QCD thermodynamics on the Grid," *Computer Physics Communications*, vol. 181, no. 10, pp. 1715–1726, October 2010. doi : 10.1016/j.cpc.2010.06.027

[263] A. J. Rubio-Montero, M. Plociennik, I. Marín-Carrión, T. Zok, M. Rodríguez-Pascual, R. Mayo-García, M. Owsiak, E. Huedo, F. Castejón, and B. Palak, "Advantages of adopting late-binding techniques through standardised interfaces for workflow managers," Poster. BoA of EGI Technical Forum 2013, Madrid, Spain, 16–20 September 2013.

[264] M. Plociennik, A. J. Rubio-Montero, T. Zok, I. Marín-Carrión, M. Rodríguez-Pascual, R. Mayo-García, F. Castejón, E. Huedo, M. Owsiak, and B. Palak, "OGSA-BES connector for Kepler to remotely use the GridWay meta-scheduler," Poster. BoA of EGI Community Forum 2013, Manchester, United Kingdom, 8–12 April 2013.

[265] E. . García, A. Saracibar, S. Gómez-Carrasco, and A. Lagana, "Modeling the global potential energy surface of the n + n2 reaction from ab initio data," *Physical Chemistry Chemical Physics*, vol. 10, pp. 2552–2558, March 2008. doi : 10.1039/B800593A

[266] D. Posada and K. A. Crandall, "MODELTEST: testing the model of DNA substitution." *Bioinformatics*, vol. 14, no. 9, pp. 817–818, October 1998. doi : 10.1093/bioinformatics/14.9.817

[267] D. Darriba, G. L. Taboada, R. Doallo, and D. Posada, "jModelTest 2: more models, new heuristics and parallel computing," *Nature Methods*, vol. 9, no. 8, p. 772, July 2012. doi : 10.1038/nmeth.2109

[268] S. Guindon and O. Gascuel, "A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood," *Systematic Biology*, vol. 52, no. 5, pp. 696–704, October 2003. doi : 10.1080/10635150390235520

[269] F. Ronquist and J. P. Huelsenbeck, "MrBayes 3: Bayesian phylogenetic inference under mixed models," *Bioinformatics*, vol. 19, no. 12, pp. 1572–1574, August 2003. doi : 10.1093/bioinformatics/btg180

[270] F. Abascal, R. Zardoya, and D. Posada, "ProtTest: selection of best-fit models of protein evolution," *Bioinformatics*, vol. 21, no. 9, pp. 2104–2105, May 2005. doi : 10.1093/bioinformatics/bti263

[271] D. Posada and T. R. Buckley, "Model Selection and Model Averaging in Phylogenetics: Advantages of Akaike Information Criterion and Bayesian Approaches Over Likelihood Ratio Tests," *Systematic Biology*, vol. 53, no. 5, pp. 793–808, October 2004. doi : 10.1080/10635150490522304

[272] D. Darriba, G. L. Taboada, R. Doallo, and D. Posada, "ProtTest 3: fast selection of best-fit models of protein evolution," *Bioinformatics*, vol. 27, no. 8, pp. 1164–1165, April 2011. doi : 1093/bioinformatics/btr088

[273] J. Abraham, P. Abreu, M. Aglietta, C. Aguirre, D. Allard, I. I. Allekotte, J. Allen, and e. a. P. Allison, "Observation of the Suppression of the Flux of Cosmic Rays above 4×10^{19}eV," *Phys. Rev. Lett.*, vol. 101, p. 061101, Aug 2008. doi : 10.1103/PhysRevLett.101.061101

[274] M. Nagano, K. Kobayakawa, N. Sakaki, and K. Ando, "Photon yields from nitrogen gas and dry air excited by electrons," *Astroparticle Physics*, vol. 20, no. 3, pp. 293 – 309, December 2003. doi : 10.1016/S0927-6505(03)00192-0

[275] F. Jansen, D. Lumb, B. Altieri, J. Clavel, M. Ehle, C. Erd, C. Gabriel, M. Guainazzi, P. Gondoin, R. Much, R. Muñoz, M. Santos, N. Schartel, D. Texier, and G. Vacanti, "XMM-Newton observatory (I. The spacecraft and operations)," *Astronomy & Astrophysics*, vol. 365, no. 1, January 2001.

[276] C. Gabriel, M. D. andD. J. Fyfe, J. Hoar, A. Ibarra, and E. Ojero, "The XMM-Newton SAS - Distributed Development and Maintenance of a Large Science Analysis System: A Critical Analysis," in *Astronomical Data Analysis Software and Systems (ADASS) XIII*, ser. ASP Conference Proceedings, vol. 314, Strasbourg, France, 12-15 October, 2003 2004, pp. 759–763.

[277] A. Ibarra, D. Tapiador, E. Huedo, R. Montero, C. Gabriel, C. Arviset, and I. Llorente, "On-the-fly XMM-Newton Spacecraft Data Reduction on the Grid," *Scientific Programming*, vol. 14, no. 2, pp. 141–150, Jaunary 2006. doi : 10.1155/2006/739583

[278] T. Böhlen, F. Cerutti, M. Chin, A. Fassò, A. Ferrari, P. Ortega, A. Mairani, P. Sala, G. Smirnov, and V. Vlachoudis, "The FLUKA Code: Developments and Challenges for High Energy and Medical Applications," *Nuclear Data Sheets*, vol. 120, pp. 211–214, June 2014. doi : 10.1016/j.nds.2014.07.049

[279] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, F. Behner, L. Bellagamba, J. Boudreau, L. Broglia, A. Brunengo, H. Burkhardt, S. Chauvie, J. Chuma, R. Chytracek *et al.*, "Geant4 - a simulation toolkit," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, no. 3, pp. 250 – 303, July 2003. doi : 10.1016/S0168-9002(03)01368-8

[280] C. Nührenberg, R. Hatzky, S. Sorge, F. Castejón, and J. Nührenberg, "Global itg turbulence in screw-pinch geometry," in *15th Stellarator Workshop and the IAEA-TM on Stellarator Theory and Innovative Concepts*, Madrid, Spain, 10–11, October 2005.

[281] V. Tribaldos, "Monte Carlo estimation of neoclassical transport for the TJ-II stellarator," *Physics of Plasmas*, vol. 8, pp. 1229–1239, 2001. doi : http://dx.doi.org/10.1063/1.1353812

[282] J. S. Tolliver, "Bounce-averaged Monte Carlo energy and pitch angle scattering operators," *Phys. Fluids*, vol. 28, pp. 1083–1089, 1985.

[283] F. Castejón, L. A. Fernández, J. Guasp, V. Martin-Mayor, A. Tarancón, and J. L. Velasco, "Ion kinetic transport in the presence of collisions and electric field in TJ-II ECRH plasmas," *Plasma Phys. Control. Fusion*, vol. 49, p. 753, April 2007. doi : 10.1088/0741-3335/49/6/005

[284] M. Vélez, J. I. Martín, J. E. Villegas, A. Hoffmann, E. M. González, J. L. Vicent, and I. K. Schuller, "Superconducting vortex pinning with artificial magnetic nanostructures," *Journal of Magnetism and Magnetic Materials*, vol. 320, no. 21, pp. 2547 – 2562, November 2008. doi : 10.1016/j.jmmm.2008.06.013

[285] E. Solano, J. A. Rome, and S. P. Hirshman, "Study of transport in the flexible heliac TJ-II," *Nucl. Fusion*, vol. 28, no. 1, pp. 157–168, 1988. doi : 10.1088/0029-5515/28/1/013

[286] A. Rodríguez-Yunta, "Neoclassical Studies in Heliac TJ-II," in *8th International Stellarator Workshop*. Kharkov, USSR: International Atomic Energy Agency (IAEA), Vienna, May 1991, pp. 69–73.

[287] K. C. Shaing and S. A. Hoking, "Neoclassical transport in a multiple-helicity torsatron in the low-collisionality $(1/\nu)$ regime," *Phys. Fluids*, vol. 26, pp. 2136–2139, 1983.

[288] H. Howe, "Physics models in the toroidal transport code proctr," Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, Tech. Rep., August 1990.

[289] D. Boucher, "Études et modelisation du transport de l'énergie et des particules dans un plasma de fusion thermonucléaire contrôlée. Aplication au transport anormal et aux conditions de fonctionnement du tokamak," Ph.D. dissertation, École Polytechnique, Paris, France, 1992.

[290] J. García-Olaya, "Study of electron heat transport in LHD and TJ-II," Ph.D. dissertation, Universidad Politecnica de Cataluña, 2006.

[291] J. Fontanet-Sáez, "Simulación de plasmas de dispositivos de fusión por confinamiento magnético tipo tokamak y stellarator. Validación experimental y aplicación al estudio del Heliac Flexible TJ-II," Ph.D. dissertation, Universidad Politecnica de Cataluña, 2001.

[292] S. P. Hirshman, K. C. Shaing, W. I. van Rij, C. O. Beasley Jr., and E. C. Crume Jr., "Plasma transport coefficients for nonsymmetric toroidal confinement systems," *Phys. Fluids*, vol. 29, pp. 2951–2959, 1986.

[293] Y. Ogawa *et al.*, "Analysis of neoclassical transport in the banana regime with the DKES code for the large helical device," *Nuclear Fusion*, vol. 32, pp. 119–132, 1992.

[294] S. P. Hirshman, K. C. Shaing, and W. I. van Rij, "Consequences of time-reversal symmetry for the electric field scaling of transport in stellarators," *Pys. Rev. Lett.*, vol. 56, pp. 1697–1699, 1986.

[295] H. Sugama and S. Nishimura, "How to calculate the neoclassical viscosity, diffusion, and current coefficients in general toroidal plasmas," *Phys. Plasmas*, vol. 9, p. 4637, 2002. doi : 10.1063/1.1512917

[296] J. L. Velasco, F. Castejón, and A.Tarancón, "Finite orbit width effect in ion collisional transport in TJ-II," *Phys. Plasmas*, vol. 16, p. 052303, May 2009. doi : 10.1063/1.3126583

[297] A. Bustos, F. Castejón, L. Fernández, J. García, V. Martin-Mayor, J. Reynolds, R. Seki, and J. Velasco, "Impact of 3D features on ion collisional transport in ITER," *Nucl. Fusion*, vol. 50, p. 125007, 2010. doi : 10.1088/0029-5515/50/12/125007

[298] F. L. Hiton and R. D. Hazeltine, "Theory of plasma transport in toroidal confinement systems," *Reviews of Modern Physics*, vol. 48, pp. 239–308, April 1976. doi : 10.1103/RevModPhys.48.239

[299] E. A. Belli and J. Candy, "Kinetic calculation of neoclassical transport including self-consistent electron and impurity dynamics," *Plasma Physics and Controlled Fusion*, vol. 50, no. 9, p. 095010, July 2008. doi : 10.1088/0741-3335/50/9/095010

[300] C. Beidler, K. Allmaier, M. Isaev, S. Kasilov, W. Kernbichler, G. Leitold, H. Maaßberg, D. Mikkelsen, S. Murakami, M. Schmidt, D. Spong, V. Tribaldos, and A. Wakasa, "Benchmarking of the mono-energetic transport coefficients – results from the International Collaboration on Neoclassical Transport in Stellarators (ICNTS)," *Nuclear Fusion*, vol. 51, no. 7, p. 076001, June 2011. doi : 10.1088/0029-5515/51/7/076001

[301] J. L. Velasco, K. Allmaier, A. López-Fraguas, C. D. Beidler, H. Maassberg, and et al., "Calculation of the bootstrap current profile for the TJ-II stellarator," *Plasma Phys. Control. Fusion*, vol. 53, no. 11, pp. 115 014–115 029, Oct. 2011. doi : 10.1088/0741-3335/53/11/115014

[302] H. Yamada, J. Harris, A. Dinklage, E. Ascasibar, F. Sano, S. Okamura, J. Talmadge, U. Stroth, A. Kus, S. Murakami, M. Yokoyama, C. Beidler, V. Tribaldos, K. Watanabe, and Y. Suzuki, "Characterization of energy confinement in net-current free plasmas using the extended International Stellarator Database," *Nuclear Fusion*, vol. 45, no. 12, p. 1684, November 2005. doi : 10.1088/0029-5515/45/12/024

[303] E. Ascasíbar, T. Estrada, F. Castejón, A. López-Fraguas, I. Pastor, J. Sánchez, U. Stroth, J. Qin, and the TJ-II Team, "Magnetic configuration and plasma parameter dependence of the energy confinement time in ECR heated plasmas from the TJ-II stellarator," *Nuclear Fusion*, vol. 45, no. 4, p. 276, March 2005. doi : 10.1088/0029-5515/45/4/009