# Distributed scheduling and data sharing in late-binding overlays

Antonio Delgado Peris and José M. Hernández
Scientific Computing Unit
CIEMAT
Madrid, Spain
{antonio.delgadoperis, jose.hernandez}@ciemat.es

Eduardo Huedo
Facultad de Informática
Universidad Complutense de Madrid (UCM)
Madrid, Spain
ehuedo@fdi.ucm.es

***Abstract*—Pull-based late-binding overlays are used in some of today's largest computational grids. Job agents are submitted to resources with the duty of retrieving real workload from a central queue at runtime. This helps overcome the problems of these very complex environments, namely, heterogeneity, imprecise status information and relatively high failure rates. In addition, the late job assignment allows dynamic adaptation to changes in the grid conditions or user priorities. However, as the scale grows, the central assignment queue may become a bottleneck for the whole system.**

**This article presents a distributed scheduling architecture for late-binding overlays, which addresses these scalability issues. Our system lets execution nodes build a distributed hash table and delegates job matching and assignment to them. This reduces the load on the central server and makes the system much more scalable and robust. Moreover, scalability makes fine-grained scheduling possible, and enables new functionalities like the implementation of a distributed data cache on the execution nodes, which helps alleviate the commonly congested grid storage services.**

***Keywords*—Grid and Cluster Computing; Scalable Computing; Peer-to-Peer Architectures and Networks; Reliable Parallel and Distributed Algorithms;**

## I. Introduction

Modern science needs massive computing. From astronomy to earth sciences or biochemistry, today's research produces ever increasing volumes of data that require higher and higher processing, transferring and storing capabilities. In order to fulfill their enormous computing requirements, scientists use a wide range of distributed computing infrastructures, from GPU processors and batch farms to grids, clouds and volunteer computing systems. Of those, large-scale grids for scientific computing probably constitute the most dramatic example of complex distributed computing system. They comprise heterogeneous resource centers spread across multiple administrative domains and interconnected by complex network infrastructures where low latencies are not always guaranteed.

Maybe the most representative example of this trend is the Large Hadron Collider (LHC) in Geneva, which generates tens of petabytes of data annually. These data must be timely stored and replicated to computing centers around the world for processing, analysis and further archival. To comply with these goals, the Worldwide LHC Computing Grid (WLCG) [1] was built. The WLCG is currently the world's largest computing grid. It integrates tens of thousands of computers and storage systems in hundreds of data centers worldwide.

In such a complex system, applications must deal with multiple access interfaces, highly heterogeneous execution environments, dynamic number of available resources (due to the infrastructures being shared) and routine component breakdowns and maintenance downtimes. The users of WLCG, organized as Virtual Organizations (VOs), must often confront unreliable resource information, large and varying time overheads in their computing jobs and important failure rates.

### A. Late-binding metascheduling

In order to alleviate the described problems, some WLCG VOs developed and progressively adopted a model of *late-binding metascheduling* for their computing jobs. This model has proven very succesful and has become the *de-facto* standard in the whole WLCG (see e.g. [2] and [3]). In late-binding scheduling, work is assigned to a node at the last possible moment before real execution. A VO agent is initially scheduled as a normal grid job with the main duty of *pulling* a real job from a VO task queue once landed on the execution resource. This VO agent is usually called *pilot job*. Even if designed for the grid, pilot job sytems have been lately used to successfully integrate clouds into the pools of VO resources.

There are several advantages in the use of pilot jobs. Firstly, pilots are sent to all available resources, using different interfaces, but present an homogeneous environment to the real applications. They also verify that computing resources are healthy (according to VO-specific criteria) before real tasks are run, thus significantly reducing failure rate. Finally, late binding means that pilots request tasks only when an execution slot is available. Therefore, uncertainties about available resources and waiting times are greatly decreased. By maintaining a constant pressure of pilot jobs on the system, VOs can adjust their internal priorities almost immediately.

We acknowledge the funding support provided by the Spanish funding agency SEIDI, through the grant FPA2010-21638-C02-02.

For big VOs, though, late-binding scheduling may also be a challenge. As the number of pilots and tasks increases, the process to match real jobs to pilots gets harder, leading to potential bottlenecks in the VO's queue of tasks.

### B. The Task Queue architecture

Besides job management issues, the biggest challenge faced by WLCG VOs is the handling of the huge amounts of data produced by LHC. In WLCG, resource centres (*sites*) offer storage services as first-class entities called *storage elements* (SEs). Computing tasks read input data from SEs and write resulting products to them. Even if these systems are designed to store petabytes of data and deal with hundreds of clients, they are not infinitely scalable and may suffer from performance problems if they are exposed to a relatively high number of very active clients. In fact, WLCG VOs usally set limits to the number of I/O-bounded jobs that can simultaneously run against a single SE.

In this context, a new pilot-based late-binding architecture, which added a data cache to reduce the I/O pressure on a site's SE, was presented in [4]. Intelligent job scheduling techniques were used to let jobs run on the nodes holding the required input data. The work showed the advantages of the late-binding approach in order to reduce workflow completion times and analyzed the impact of the data cache on different workflow types and SE congestion conditions. The described architecture however proved to be very demanding on the central server, the *Task Queue* (TQ). In order to maximize the cache hit ratio, the TQ must keep track of every data product in the nodes and, whenever a job request from a pilot arrives, all queued tasks must be examined. We have observed that this can lead to severe matching delays when the scale of the system (number of pilots and tasks) increases.

In the present article, we introduce a new architecture which offers several enhancements to that proposed in [4]. In the new system, pilot nodes arrange themselves into a Kademlia network [5] within a site's local area network (LAN). That is, pilots share a distributed hash table (DHT) and use it to locate and reach other nodes as well as a common key-value store. In particular, the location of cached data files is recorded in the DHT, enabling pilots to locate and retrieve the input files they need from other peer nodes, greatly increasing the effective cache hit ratio of the system.

Moreover, the new inter-pilot communication capabilities have made it possible to implement a distributed matching algorithm, by which pilots in a site collectively evaluate, rank and assign computational tasks. The TQ is freed from tracking files in the cache and performing the task matching process. This increases the scalability of the whole system and reduces the coupling of execution clusters with the central server. The latest also means fewer wide area network interactions, which becomes important when latencies are high.

The rest of this paper is structured as follows: Section II presents the problem of job scheduling and data handling in large grids and reviews related work on the subject. Section III describes the proposed architecture and Section IV provides an experimental evaluation of the whole system. Finally, Section V summarizes and presents conclusions.

## II. Problem statement and related work

Large grid scheduling traditionally considers sites as homogeneous sets of resources. At most, a site offers a few different queues with varying job length limits or different memory capacities [6]. But the number of resource groups has to be necessarily very limited to make it manageable by both users and site admins. However, in late-binding systems, the central server of a VO assigns jobs directly to individual pilots, so it can potentially consider the particular characteristics of each node (rathen than those of the site or queue). We call this *micro-scheduling*.

As indicated in [4], the Task Queue's original aim was to take advantage of the pilot jobs used in today's late-binding scheduling systems to create a data cache that could reduce the pressure on a site's SE. An effective data cache requires a high hit ratio and this is only possible if computing tasks are scheduled to the appropriate execution nodes—those holding the data they depend upon. Micro-scheduling was, thus, necessary for our goal, but, beyond that, we consider it a powerful technique, which might be used in other interesting ways. A fine-grained task matching would make it possible to select work for a pilot based on its precise characteristics, such as memory, disk space, number and power of (free) CPUs[a], attached instruments, software licenses, etc. Moreover, information about the jobs running in a pilot's node could be used to balance the number of I/O-bounded and CPU-intensive tasks so that the WN was used more efficiently.

There is however a major problem with this view: the central queue of tasks may become a bottleneck for the scheduling and execution of large-scale workloads [2], [8]. As the number of pilots increases, so does the rate of requests to be served. Moreover, a longer queue of pending tasks means a higher number of candidates to be inspected for each request. Existing pilot systems have already met this problem and have found different ways to tackle it. Tasks may be grouped and evaluated in bulk or pilots may be matched based on their site only [8]. In this case, we are effectively restricting or even giving up micro-scheduling. Another possible technique is to match pilots to tasks beforehand so that each request is tied to a particular task so no real matching has to be carried out [3]. This means that early-binding—rather than late-binding—scheduling is performed.

However, as indicated, our TQ architecture aims to offer real fine-grained late-binding micro-scheduling. Furthermore, the task assignment process used in the TQ supports not only task matching but also task ranking [4]. This means that the pilot will not get any task whose requirements it fulfills, but

[a]This seems specially interesting for WLCG VOs now that they are trying to move from single-core jobs to single-node but multi-core tasks [7].

the *best match*, according to a predefined ranking expression. Thus, semantics like *run task preferably on a pilot holding certain file* (best match) *but, otherwise, on any other pilot* (any match) can be supported. However, evaluating both the requirement and rank expression on all queued tasks for every pilot request is a CPU-intensive duty, which can lead to TQ congestion and a rise in the time spent serving each request. Notice that while a pilot waits for the assignment of a task, it is idle and it is wasting CPU time. This delay is a *scheduling overhead*, which should be reduced as much as possible. This is actually the main reason for the development of the new distributed architecture.

But even if the TQ is able to serve requests promptly, it process them one at a time, and, thus, it may not make the best assignments from a wide point of view; i.e.: considering all pilot-task mappings as a whole. To address this, our new architecture aims to maximize the sum of rank values for all task assignment (*global ranking*). Furthermore, in the data cache use case, there are occasions when the assignment policy cannot possibly improve the hit ratio. E.g., merge jobs must read several input files, in general located at different pilots, so it is impossible to find all of them in one node's local cache. The new architecture copes with this problem by having pilots retrieve files from one another, by means of the shared DHT.

Finally, another concern is that the central queue of a pilot system becomes a single point of failure. One way to deal with this is to set redundant high-availability configurations for the server. However, our new architecture is a first step towards reducing the pilots' dependency on the TQ and thus increasing the robustness of the system against TQ failures.

### A. Related work

Even if pull-based overlay architectures were proposed earlier [9], DIRAC [10] was the first pilot-based system used by a large VO in WLCG. Since then, several other VOs have adopted the same model. Examples of large-scale systems in use today are Alien [11], PanDA [12] and glideinWMS [13]. The functionalities offered by these systems and the lessons learnt by their VOs serve as a major inspiration for our work. All of them, however, rely on a centralized task matching and assignment procedure, and are, thus, subject to potential scalability problems as we have seen. Likewise, none of their architectures implement a dynamic data cache on the pilot nodes nor support inter-pilots communication capabilities.

Distributed Hash Tables (DHTs) provide reliable and scalable routing and look-up services without the need of central coordination. Nodes can use them to store key-value pairs, retrieve the value associated to a given key and locate a node given its identifier. In particular, *Kademlia* is a well-known and popular DHT system and presents some convenient characteristics based on its XOR-base metric topology [5]. Certainly, we can find previous work on the use of peer-to-peer (P2P) techniques for file location tracking, which have also inspired our distributed cache. The paradigmatic example of DHT-based data cache is memcached [14]. Another example is [15], which proposes a P2P grid replica location catalog.

There have also been prior efforts to utilize distributed techniques for job scheduling. SwinDeW-G [16] is a workflow management system where grid nodes interconnected using P2P techniques coordinate to distribute computational tasks. Likewise, [17] suggests the deployment of local schedulers on resource centres and their cooperation using a P2P network and [18] proposes to create a DHT-based logical coordination space where grid resource and workflow agents can cooperatively schedule their workflow tasks. However, the granularity of all these approaches is at the site level, thus no micro-scheduling is performed, and their P2P networks are grid-wide while our DHTs are confined to a site's LAN. Furthermore, their models assume early-binding matching (even if they expect their systems to have more reliable information than traditional grid brokering systems). Lastly, none of these works show tests of the scale that we do.

Finally, the proposed pull-based paradigm can also be seen as a generalization of the well-known master-worker pattern. Authors in [19] propose a hierarchical version of this paradigm, which can be seen as analogous to the task delegation to the site' pilots occuring in our architecture. However, this is the only resemblance point since, within their subclusters, centralized task scheduling is used.

## III. Description of the system

In order to address the challenges described in the previous section, we have designed a new, scalable, DHT-based architecture that allows us to perform micro-scheduling, increase the robustness of the system and improve the overall task to pilot assignment (better global ranking, higher cache hit ratio). The description of this system follows.

### A. File Sharing

We have used the Kademlia DHT to build a distributed data cache, with the following goals:

- Relief the Task Queue of the duty of keeping track of all the data files in the pilots cache.
- Make it possible for pilots to locate and fetch files from peer nodes when required (increase cache hit ratio).

When a pilot job is started on a WN, it first contacts the TQ to register itself. At that point, the TQ assigns it a Kademlia identifier (generated with a hash function) and gives it a list of contacts at its site. With this information, the pilot can initiate the procedure to join the corresponding network by reaching those contacts and filling some entries of its DHT routing tables. Correspondingly, it will be added to the tables of its peer nodes as necessary. The node is now part of the Kademlia network and can ask for jobs to run.

The sharing of files via the DHT works as follows: When a new file is produced, the pilot stores a key-value pair in the

DHT, where the key is the hash of the file name (or unique identifier), and the value is its location. This information is replicated on a number of nodes (typically, three) that are close to the key according to Kademlia metrics. When a pilot is assigned a job for execution, it examines its data dependencies. Every required file is first looked for in the local cache and, if not there, a DHT finding procedure is initiated with the hash of the file name as argument. Once the location of a replica is retrieved, the pilot fetches the file and adds it to its local cache. For this purpose, every pilot runs a simple file server accepting requests from other peers. All this procedure takes place before the umodified real job is started. At execution time, the job behaves as usual: it reads input data from local cache, or, upon failure, falls back to the local SE [4].

Our implementation of the Kademlia protocol introduces a few changes to the original specification. Firstly, when a pilot needs to leave the system, it sends a `Leave` message to its known contacts, so that they update their routing tables, and it republishes its key-value pairs so that the storage redundancy is kept. This contributes to a more proactive adaptation to node departures in contrast to the expiration-based behaviour of pure Kademlia. Secondly, if a `Store` message is received for an already known key, the new value is appended to the existing one. This makes it possible to associate multiple locations (replicas) to a single file, increasing the chances of later retrieval. Finally, while Kademlia authors propose to expire newest contacts first, based on the observation that tipically oldest DHT contacts remain alive longer, we apply a simple least-recently-seen expiration policy. We do so because grid pilots are expected to have a fairly constant lifetime, determined by batch system limits.

### B. Distributed task scheduling

As discussed in Section II, the heaviest duty performed by central task queues, and in particular by our original TQ, is the assignment of real jobs to pilots. The TQ can become a bottleneck for the whole scheduling system. That is why we have designed a new DHT-based job assignment architecture, with the following aims:

- Reduce load on the TQ; avoid it becoming a bottleneck.
- Reduce the interactions of the pilots with the TQ, increasing their autonomy.

In our new architecture, there is a representative per site, called master pilot, or, simply, *master*. Since every pilot needs to register with the TQ on wake-up, the TQ can assign the master role to a site's first registered pilot and inform the following ones about it. When the master pilot dies, a simple election process is carried out to choose the pilot with lowest identifier and the TQ is informed about the result.

The master contacts the TQ and requests tasks for its site. The TQ needs only match those tasks with no data dependency (able to run on any site) or depending on data held at the site's SE. The description and the requirement of all the matching tasks are passed to the master and this broadcasts the information to the rest of pilots in the site (*workers*). The workers filter and rank the tasks and send the results back to the master, which then assigns a task to each pilot, trying to maximize the sum of individual ranks, and informs both the TQ and the pilots about the mapping. The pilots proceed to download the task that was assigned to them, fetch the necessary input files and run the job. When the task is finished, the pilot informs the master about it and uploads the resulting job report to the TQ. Figure 1 shows a simplified diagram of the interactions among the different components of the system.

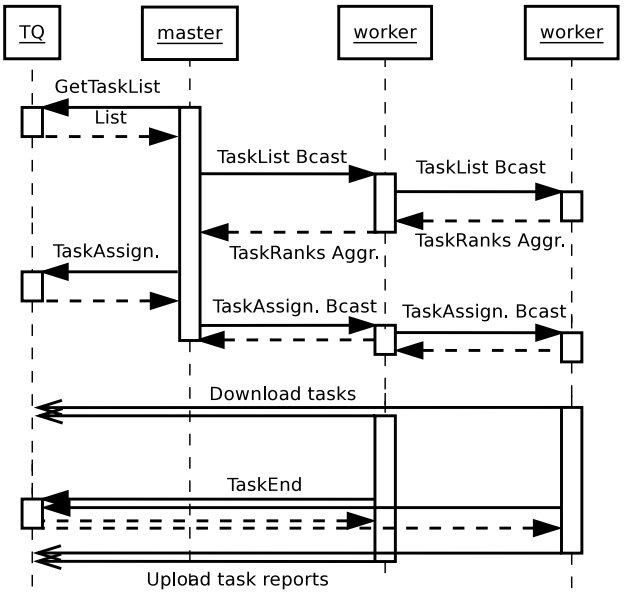


Figure 1. Simplified diagram of the distributed scheduling process.

From this description, we can see that the task of the TQ is much lighter now. Firstly, it only receives job requests from one node per site. Secondly, these require a simple (bulk) selection based on a single criterion (site) and not a complex matching process. Furthermore, since the master node is in permanent contact with the workers, it can also inform about non-responding or dead nodes, and therefore periodic heartbeat messages from workers are not needed anymore (just as the DHT meant the removal of the file tracking reports). At this moment, the only worker messages requiring TQ processing occur at registration, shutdown and task completion times. Not only have we greatly reduced the number of requests the TQ must serve but we have also eliminated those involving costly computation. Moreover, it should also be trivial to avoid the workers' task end messages by letting the master inform of job completions in bulk. Notice that this would defer the moment when the TQ notices a task is finished (and, thus, the scheduling of dependent jobs). The impact of this delay in the scheduling overhead remains to be studied. Finally, let us note that although the workers need to download real jobs and upload completion reports, these interactions entail no logical processing at the TQ other than the pure file serving/storing. In other words, the TQ does not perform any state change so

these duties could be just performed by a separate file server.

The intra-site matching and ranking process depends on the dissemination of the list of runnable tasks from the master to the workers (`TaskList` message) and the collection of results by the master. This required a major enhancement of the Kademlia protocol to add support for broadcast and aggregation capabilities, which resulted in an in-depth study of the broadcast problem, described in [20]. In the current implementation, each node divides the space it must broadcast to in eight regions and forwards the message at hand to one of its contacts in each region. This means that, no matter the size of the network, the master will never need to send messages to (or receive from) more than eight workers. For those broadcasts that require an aggregated reply, each receiving node keeps a table of expected replies and a pointer to its *parent node* (the one sending the message to it). When all the replies are received, or after some timeout, dependent on the node's depth in the broadcast tree, the aggregated result is sent back to the parent. Late messages are simply discarded but this is not too serious: a pilot not getting a task in a given round just needs to wait for the next call to come.

### C. Task matching and ranking

Even if, with the new architecture, pilots can fetch required files from peer nodes, it is still better for tasks to land on pilots holding the files they need, since reading from the local cache on disk should be cheaper than through the network. In more general terms, we set the following goal for our architecture:

- Produce a global ranking at least as good as in the previous architecture (or better).

The procedure to assign tasks to pilots is in practice not very different from that used in the centralized scheduling. As discussed in Section II, the task requirement and ranking expressions are the key to select the best job for each pilot. In fact, all the information the TQ uses for the evaluation of these expressions is the description of the task and the characteristics of the pilot. By broadcasting the task descriptions, the individual pilots are able to match and rank the tasks with exactly the same results as when the TQ did it centrally. Since this is done on a per pilot basis, even complex matching and ranking functions can be used with no scalability problem.

Notice, however, that with the old scheduling procedure, the TQ would match one pilot at a time, while, in the new model, the master receives information from all pilots in a site and then performs a collective assignment, searching the best global result. Indeed, with the received rank values per pilot, the master builds a rank matrix where the row and column of each value corresponds to the evaluated task and the reporting pilot respectively. Now the master needs to maximize the sum of matrix elements with the constraint that only one element per row and per column can be chosen (a pilot can run only one task and a task can be run only by one pilot). This is known to be an NP-complete problem but we can choose to accept a suboptimal solution in order to reduce the resolution time. In particular, we have implemented the following algorithm:

1) Look for the maximum value of the matrix.
2) Associate the corresponding pilot-task pair and remove the row and column from the matrix.
3) If the matrix is empty, finish.
4) Go back to the first step.

Since we are now manipulating a numerical matrix, this algorithm can be performed very quickly and produces quite satisfactory results. If we described the old procedure of matching one pilot at a time in similar terms, we would be replacing the first step with the following one:

1) Look for the maximum value of the first column.

It is easy to see that the results obtained with the distributed architecture will in general be better than those got with the centralized one. In fact, this will be reflected by the results in our tests, in particular in what regards the cache hit ratio, as discussed in the next section.

## IV. Experimental results

In order to evaluate the new architecture, a testbed capable of simultaneously running more than 1,500 pilots was built at the CIEMAT institute, in Madrid. Two sets of resources were used in the tests. The first comprises slots from an infiniband-based parallel cluster facility, while the second one includes the nodes of the institute's WLCG grid site. In our configuration, the two sets are viewed as two different sites.

With the aim of comparing the scalability of the old and new architectures, the same workflow types used in [4] have been run. These are inspired by real data-driven WLCG workflows, which can benefit by the use of a data cache. Tasks are chained in two steps. Every task produces a single data file but those of the second step consume the data generated at the first one. The workflow types are *serial chain* (each data file is read by a different job), *splitting* (each data file is read by two jobs) and *merging* (each job consumes two files). Unless otherwise indicated, all the described experiments show the results from three independent runs of each workflow type.

The independent variables considered in the experiments are the testbed size and the used architecture. Regarding the size, an increasing number of pilots (approximately 100, 500, 1500 and 1500; including the two resource sets) and tasks (300, 1.5k, 5k and 10k, respectively) was used. As for the architecture, the first one (*pre-DHT*) is that presented in [4]: the central TQ keeps track of all data files and also performs the task matching. The second one (*centralized*) represents an intermediate stage, in which the DHT has been introduced and file location is tracked by the pilots, but matching is still centralized on the TQ (for this, the pilot must inform about the contents of its cache in its requests). Finally, our newly proposed architecture is labeled *distributed*. In this case, files are tracked via the DHT and task matching is performed by the pilots themselves using a distributed algorithm.

### A. Pressure on Task Queue and scheduling overhead time

The first set of results addresses the scalability of the three architectures under study. The presented measurements include the number of requests served by the TQ, the scheduling overhead and the total worfklow time, for each of the configurations and testbed sizes. For all presented plots, the error bars represent the standard deviation of the mean.

Figure 2 shows the amount of requests received by the Task Queue during a workflow run. The value increases almost linearly with the number of tasks in the workflow. In the pre-DHT configuration, each job causes around 12 TQ requests. Pilots contact the TQ for both task retrieval and file tracking. When the DHT is introduced but the task matching is kept centralized, the number of requests decreases moderately (to 9 per job) because no messages are sent to track the creation and deletion of files. Finally, when the new distributed matching process is used, the master pilot takes care of all task matching and heartbeat interactions with the TQ and thus the number of requests decreases to around 5 per task. Notice that, as described in Section III, with this architecture, not only are there fewer interactions, but these are also lighter requests.

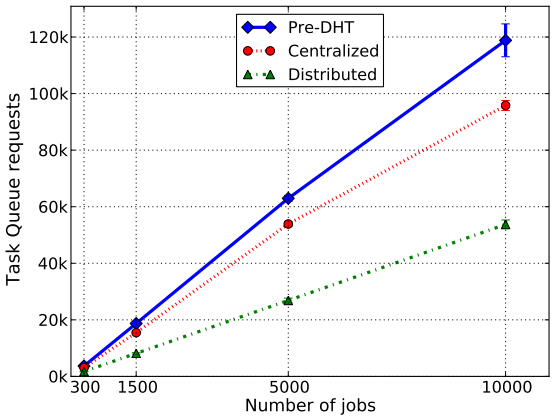


Figure 2. Number of TQ requests per architecture and testbed size.

Relieving the TQ of the costly processing duties is the key to make it scalable. Figure 3 displays the delay between a job end in a pilot and the next job start. Since, in our tests, workflow tasks are always ready to be served to pilots (or otherwise there are no more tasks to be run), this delay is a measure of the scheduling overhead. We find that both pre-DHT and centralized architectures behave nicely for small and medium scales (around 5 seconds of overhead) but show worse performance (and much higher dispersion) when the number of pilots and tasks increase. The reason is clear: the TQ cannot keep pace with the increasing request rate (more pilots), especially because each request demands heavier processing (more queued tasks to review).

The distributed architecture however scales nicely and offers almost constant inter-tasks delay. For smaller testbeds, this

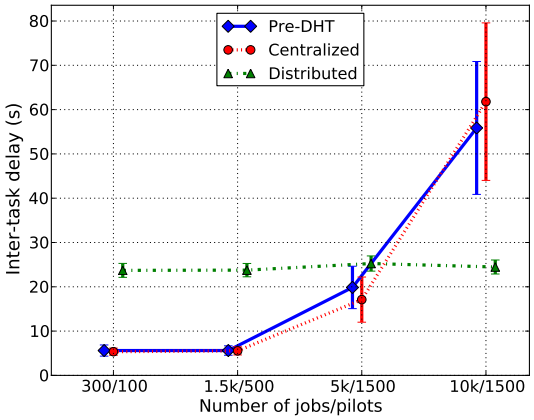


Figure 3. Inter-tasks delay per architecture and testbed size.

interval is somewhat longer than in the previous configurations but remains low even for the largest testbed. In fact, the delay is consistent with the period between `TaskList` messages sent by the master pilot (20 s). Even if this number seems significant in our tests, where the average job duration is 3.3 minutes, it would be negligible for an average task duration in the order of hours (as expected for WLCG). Certainly, an increased job length would also result in a reduced request rate, thus alleviating the problems of centralized matching. Notice however that, first, this would not reduce the long queue of tasks to review on each request, and, more importantly, the average request rate is directly proportional to the number of pilots but the frequency of the `TaskList` messages is not. The DHT broadcast structure allows the master to communicate with a high number of nodes without any increase in the number of messages it has to deal with. Moreover, the expected delay for the replies is of logarithmic growth [20].

The same scaling pattern can be appreciated in Figure 4, which compares workflow turnaround times for the different cases. The plot shows softer differences than those of inter-tasks delay for small testbed sizes because job execution times are basically identical for all configurations. However, as the number of pilots and jobs increases, the results for pre-DHT and centralized configurations worsen as quickly as before. This is due to the inter-tasks delay telling only half of the story of the TQ congestion. For the complete picture, we must refer to Figure 5, which summarizes the most relevant data for an individual serial-chain workflow run of 10,000 jobs and 1500 pilots, using the pre-DHT architecture.

The upper plot of Figure 5 shows the number of running jobs versus time. When the workflow starts, all the pilots in the sytem contact the TQ to request their first task and download it at barely the same time. The TQ becomes overloaded and the time spent on each request rises. It takes the TQ almost 10 minutes to serve them all. Only then, can the pilots retrieve and start their real jobs. This initial congestion is not reflected by the inter-tasks delay metric because only the interval *between*

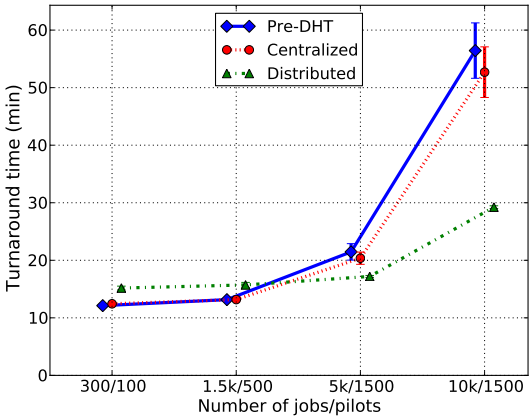


Figure 4. Turnaround workflow time per architecture and testbed size.

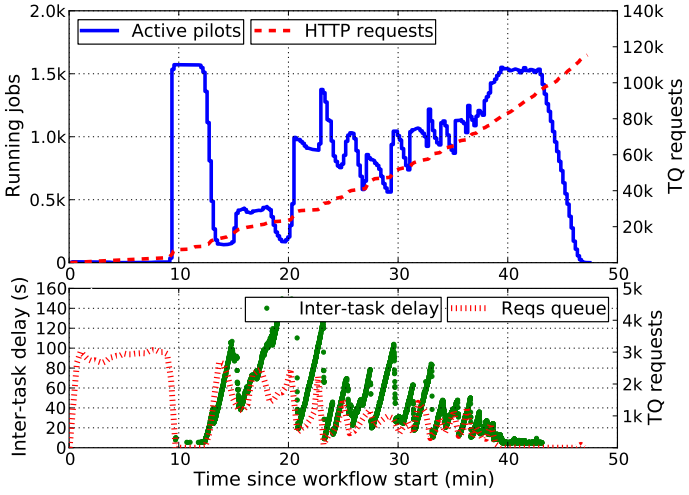


Figure 5. Slot occupancy and cumulative TQ requests (top); inter-tasks delay and requests queue length (bottom); for the pre-DHT architecture.

tasks is measured and these are the very first tasks on each pilot. The lower plot displays the correlation of the TQ activity with the inter-tasks delay and the length of the queue of pending requests. It also gives a feeling of the unpredictability of the job scheduling overhead with this configuration.

For comparison, Figure 6 shows a run of exactly the same characteristics of the previous one but using the distributed architecture. This time, the number of running jobs and the inter-tasks delay follow a much more regular (and healthy) pattern and the queue of pending requests at the TQ is consistently kept very low (almost empty) during the whole lifetime of the workflow.

### B. Cache hit ratio

Since the original motivation for the TQ architecture was to reduce the pressure put on SEs, it is also important to evaluate how the new architecure behaves in regard to the ratio of cache

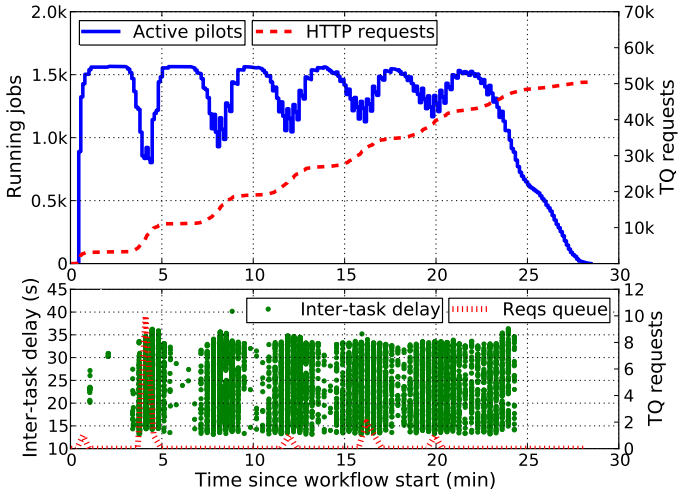


Figure 6. Slot occupancy and cumulative TQ requests (top); inter-tasks delay and requests queue length (bottom); for the distributed architecture.

to total reads (files not found in the cache are read from the SE) and also the ratio of files read from the local disk cache.

Figure 7 displays the cache hit ratio for every architecture and testbed size. The configurations using the DHT have an almost perfect record of cache reads for every case because jobs can retrieve files from other nodes owning them. With the pre-DHT architecture however, the hit ratio is considerably lower since jobs can access only files on its own disk). It is interesting to note that, in this configuration, the hit ratio increases for larger testbeds. For brevity's sake, we will not get into details here, but we are aware of some race conditions that may cause a cache miss on the first second-step task that is run on a pilot. Since in the large testbeds, the ratio of tasks to pilots is higher, these misses have a smaller relative impact.

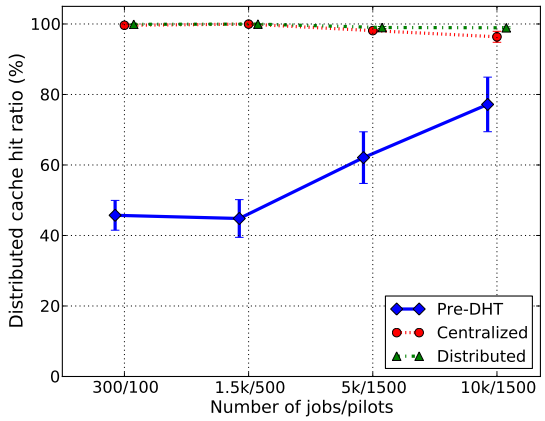


Figure 7. Distributed cache hit ratio per architecture and testbed size.

Figure 8 shows the ratio of jobs reading from the local cache on disk. The pre-DHT configuration behaves exactly as in Figure 7 since no DHT reads are possible in this case.

This time, however, the centralized architecture obtains similar results. This is natural since both configurations use the same matching algorithm. For the distributed configuration, though, the hit ratio is much better and it does not vary significantly with the testbed size. As discussed in Section III-C, in this case, the master matches all known tasks and pilots at regular intervals and it can thus perform a better global matching than when serving requests one at a time. The race conditions described earlier for the first set of second-step jobs are less significant with this algorithm and the results are similar for workflows with either low or high number of tasks.

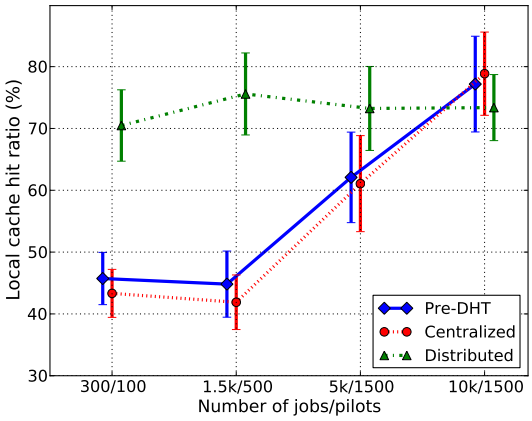


Figure 8. Local cache hit ratio per architecture and testbed size.

Notice also that the dispersion of results for every case is relatively high. This is due to the lower outcome of the merging workflows in comparison to those obtained with serial chain and splitting (as indicated in Section II).

## V. Conclusions

We have presented a new pull-based late-binding overlay architecture that, by creating a DHT system among pilot nodes, is able to offer an effective data cache and a distributed scheduling mechanism, which is far more scalable than its centralized alternative. The new system is also more robust: there is less interaction between the sites and the central queue, which has become simpler and less loaded. We have shown that the architecture produces limited and predictable scheduling overheads, even at large scales, and that this leads to shorter workflow completion times. We have also verified that by performing bulk task matching at regular time intervals, the global assignment rank can be increased; in particular, the hit ratio of our data cache rises, even for small workloads. Furthermore, DHT-based file sharing services turn the local cache into a distributed one, with an almost perfect hit score.

As future work, we plan to enhance the robustness of the system against Task Queue downtimes or network disruptions. We also intend to study the suitability (and gained benefits) of using our architecture in resource centres with no local storage services available. Finally, we generally aim to deepen our understanding of how micro-scheduling and inter-pilots communications can be used to improve the execution of massive workloads in large, complex, distributed computing infrastructures.

## References


[1] "WLCG: Worldwide lhc computing grid," 2014. [Online]. Available: http://wlcg.web.cern.ch

[2] S. K. Paterson and A. Tsaregorodtsev, "DIRAC optimized workload management," in *Journal of Physics: Conference Series*, vol. 119, no. 6. IOP Publishing, 2008, p. 062040.

[3] T. Maeno, K. De *et al.*, "Evolution of the ATLAS PanDA production and distributed analysis system," in *Journal of Physics: Conference Series*, vol. 396, no. 3. IOP Publishing, 2012, p. 032071.

[4] K. Hasham, A. Delgado Peris *et al.*, "CMS workflow execution using intelligent job scheduling and data access strategies," *Nuclear Science, IEEE Transactions on*, vol. 58, no. 3, pp. 1221–1232, 2011.

[5] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS '01)*. London, UK: Springer-Verlag, 2002, pp. 53–65.

[6] P. Andreetto, S. Andreozzi *et al.*, "The gLite workload management system," in *Journal of Physics: Conference Series*, vol. 119, no. 6. IOP Publishing, 2008, p. 062007.

[7] J. Hernández, D. Evans, and S. Foulkes, "Multi-core processing and scheduling performance in cms," in *Journal of Physics: Conference Series*, vol. 396, no. 3. IOP Publishing, 2012, p. 032055.

[8] D. Bradley, T. St Clair *et al.*, "An update on the scalability limits of the condor batch system," in *Journal of Physics: Conference Series*, vol. 331, no. 6. IOP Publishing, 2011, p. 062002.

[9] C. Pinchak, P. Lu, and M. Goldenberg, "Practical heterogeneous placeholder scheduling in overlay metacomputers: Early experiences," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2002, pp. 205–228.

[10] A. Tsaregorodtsev, V. Garonne *et al.*, "DIRAC–distributed infrastructure with remote agent control," in *Proc. of CHEP2003*, 2003.

[11] P. Saiz, L. Aphecetche, P. Bunčić, R. Piskač, and J.-E. Revsbech, "Alien–ALICE environment on the GRID," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 502, no. 2, pp. 437–440, 2003.

[12] T. Maeno, "PanDA: distributed production and distributed analysis system for ATLAS," in *Journal of Physics: Conference Series*, vol. 119, no. 6. IOP Publishing, 2008, p. 062036.

[13] I. Sfiligoi, "glideinWMS–a generic pilot-based workload management system," in *Journal of Physics: Conference Series*, vol. 119, no. 6. IOP Publishing, 2008, p. 062044.

[14] B. Fitzpatrick, "Distributed caching with memcached," *Linux journal*, vol. 2004, no. 124, p. 5, 2004.

[15] A. Chazapis, A. Zissimos, and N. Koziris, "A peer-to-peer replica management service for high-throughput grids," in *Parallel Processing, 2005. ICPP 2005. International Conference on*. IEEE, 2005, pp. 443–451.

[16] Y. Yang, K. Liu, J. Chen, J. Lignier, and H. Jin, "Peer-to-peer based grid workflow runtime environment of swindew-g," in *e-Science and Grid Computing, IEEE International Conference on*. IEEE, 2007, pp. 51–58.

[17] J. Cao, O. M. Kwong, X. Wang, and W. Cai, "A peer-to-peer approach to task scheduling in computation grid," in *Grid and Cooperative Computing*. Springer, 2004, pp. 316–323.

[18] M. Rahman, R. Ranjan, and R. Buyya, "Cooperative and decentralized workflow scheduling in global grids," *Future Generation Computer Systems*, vol. 26, no. 5, pp. 753–768, 2010.

[19] J. Berthold, M. Dieterle, R. Loogen, and S. Priebe, "Hierarchical master-worker skeletons," in *Practical Aspects of Declarative Languages*. Springer, 2008, pp. 248–264.

[20] A. Delgado Peris, J. M. Hernndez, and E. Huedo, "Evaluation of the broadcast operation in Kademlia," in *IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, G. Min, J. Hu *et al.*, Eds. IEEE Computer Society, 2012, pp. 756–763.