Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 230 – 239

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

SANComSim: A Scalable, Adaptive and Non-intrusive framework to optimize performance in Computational Science applications

Alberto Núñez[a], Rosa Filgueira[b] and Mercedes G. Merayo[a]

[a]Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain
[b]School of Informatics, University of Edinburgh, Scotland
email: alberto.nunez@pdi.ucm.es, rosa.filgueira@ed.ac.uk, mgmerayo@fdi.ucm.es

Abstract

Parallel processing has become the most common solution for developing and executing scientific computing applications. Actually, the best way to obtain good performance ratios is to exploit parallelism in both processing and communications. Although the study of computational performance has historically involved CPU power, currently the CPU is not the only concern in the overall performance. Due to the underlying design of parallel applications, communication networks play a very important role in the field of computational science. Despite the fact that networks used in multicore clusters are fast and have low latency, the amount of transferred data may cause a bottleneck in the communication system, as communication-intensive, parallel applications spend a significant amount of their total execution time exchanging data between processes. Moreover, in most cases, several users are executing different parallel applications at the same time in the cluster.

In this paper we present SANComSim, a Scalable, Adaptive and Non-intrusive framework, based on simulation techniques, for optimizing the performance of the network system to execute complex applications. The main objective of this framework is to apply run-time compression, to reduce the data sent through the network, in order to increase the overall system performance. The main features of SANComSim are: adaptability, to dynamically adapt to the current state of the system; portability, the framework is neither focused on a specific programming language nor a platform; non-intrusive, since this framework is based on simulation techniques, which does not require exclusive access of the entire cluster system; scalability, any parallel application, independently of the number of processed and computing nodes, can use this framework to improve performance in cluster systems.

Keywords: Parallel computing optimizations, Simulation and modelling, Compression techniques

1. Introduction

Parallel processing has become the most common solution for developing and executing scientific computing applications. Generally, this kind of application is usually both computation and data intensive. Therefore, the best way to obtain good performance ratios is to exploit parallelism in both processing and communications. The current trend is to use multicore clusters in order to increase the computation capability, thus allowing an increase in the number of processes per application. Examples of these applications can be found in many fields of computational science like MRI scan data [1], molecular dynamics [2], simulations [3] and mathematics [4].

*Corresponding author. Tel.: +34-91-394-7646 ; fax: +0-000-000-0000 .
E-mail address: alberto.nunez@pdi.ucm.es.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.186

Due to its high cost-effectiveness, commodity clusters have been the main resource to deploy computational applications. As an example, in the most recent survey (November 2012) of the fastest 500 computers in the world [5], 82.2% are clusters, where a great part of those are medium-size clusters.

Computational performance has historically involved CPU power. However, the CPU is not the only concern in the overall performance of complex systems. The network also plays an important role in performance. Currently, although the number of CPU cores per computing node has increased considerably, the network bandwidth has not evolved as fast as the CPU power. For instance, having in mind the TOP500 list [5] in November 2010, 50.1% of these systems used quad-core CPU, and 25% use 6-core processors. Two years later, in November 2012, 39,2% used 16-core processors and 32.9% used 8-core processors. This means a double up of CPU cores per node in two years. However, the network has not experimented such growth. In November 2010, 43,8% of the top 500 systems used Ethernet Gigabit, and only 29% used Infiniband. Two years later, in November 2012, 31.8% used Ethernet Gigabit and 21,2% used Infiniband, which means that practically the same percentage of systems used the same network architecture.

Due to the underlying design of parallel applications, communication networks play a very important role in the field of computational science. Despite the fact that networks used in multicore clusters are fast and have low latency, the amount of transferred data may cause a bottleneck in the communication system, as communication-intensive, parallel applications spend a significant amount of their total execution time exchanging data between processes. Moreover, in most cases, several users are executing different parallel applications at the same time in the cluster. Although each user can request exclusive access of a specific number of computing nodes, the network is a shared resource used by all the users in the cluster. Currently there is a wide range of solutions to alleviate this network issue in cluster systems. A possible solution consists in compressing the data sent through the network to reduce transmission times, therefore, alleviating congestion in the communication system. However, although the concept is very simple, applying compression techniques in the network traffic efficiently to improve performance is a very involved and complex task. The main issue is that it is not ensure compressing data always provides better performance. In some cases, the time spent to compress-send-uncompress data is greater than the one needed to send the data without applying compression. Consequently, there are several aspects that have to be considered for applying compression or not: the size of the message, the type of data, the network usage at the time of send the message and the compression algorithm to be applied.

In this paper we present an adaptive and scalable framework, based on simulation techniques, for optimizing the performance of the network system to execute computational parallel applications. The main idea of this framework is to apply run-time compression to reduce the data sent through the network, and then increasing the overall system performance. Moreover, this framework is dynamically adapted to the current state of the system. Consequently, different aspects are considered before taking a decision of compressing a message. They involved the percentage of network utilization, the type of data sent, the size of each message and the compression algorithm. The main contributions of our proposal are summarized below:

- Our framework is both platform and application independent: it can be used in any cluster and for any application. Since it is based on heuristics depending of different parameters, this framework can be dynamically adapted to the considered scenario.
- In contrast to other existing methods, our framework does not require exclusive access to the cluster for obtaining measurements about the network characteristics (latencies, bandwidth, etc.). This is achieved by using a simulation platform to obtain accurate measurements of different network scenarios. This is very useful in clusters where different users are executing applications in the cluster.

The rest of the paper is structured as follow: Section 2 presents related work. Section 3 describes the main architecture of the proposed framework. Section 4 presents a set of performance experiments for validating our framework. Finally, Section 5 presents conclusions and suggests some lines for future work.

2. Related work

Compression [6] has been used in a wide spectrum of research fields, such as minimizing disk utilization, decreasing bandwidth consumption on networks, reducing energy consumption in hardware and in the multimedia domain, where images and movies are compressed to save disk space and network transmission. In this

paper we focus on applying compression techniques to increase the network performance in scientific computing applications.

We study the effect of compression in data exchanged among processes using the communication network. It is important to note that this paper does not consider the I/O system, which is mainly focused on storage. On the contrary, we investigate the effects of data compression in inter-process communications. We show that, in some cases, applying the right run-time compression algorithm improves network end-to-end transmission time, network bandwidth, and consequently the overall system performance.

Many studies have been done on the effects of compression to minimise the volume of communications between processes. COMPASSION [7] is a parallel I/O run-time system which includes chunking and compression for irregular applications. The LZO algorithm is used for fast compression and decompression. The main drawback of this approach is the lack of different compression algorithms. Moreover, it is only used for the I/O part of irregular applications.

PACX-MPI (PArallel Computer eXtension to MPI) [8, 9] is an on-going project of the HLRS, Stuttgart. It enables an MPI application to run on a meta-computer consisting of several, possibly heterogeneous machines, each of which may itself be massively parallel. Compression is used for TCP message exchange among different systems in order to increase bandwidth, but a fixed compression algorithm is used and compression is not used for messages within single sub-system. cMPI [10, 11] has similar goals o those of PACX-MPI, namely to enhance the performance of inter-cluster communication with a software-based data compression layer. Compression is added to all communication, so it does not offer any flexibility as to how to configure when and how to use compression.

CoMPI [12] was the first work in which a compression library was fully integrated into MPICH. CoMPI is based on run-time compression of the MPI messages exchanged among applications. The user can choose the compression algorithm from a pool of algorithms, and all the communications will be compressed with the same algorithm. The problem with this approach is that the user cannot always select the most suitable compression algorithm, and compression is always turned on by default.

Dynamic-CoMPI [13] presents an optimization of MPI communications, called Dynamic-CoMPI, which uses different techniques in order to reduce the impact of communications and non-contiguous I/O requests in parallel applications. These techniques are independent of the application and complementary to each other. The technique called Adaptive-CoMPI is based on run-time compression of MPI messages exchanged by applications. This approach presents difference issues. First, this technique is very intrusive because it requires an initial execution to create information about the Network-behavior. The Network-behavior is based on computing the bidirectional bandwidth between every pair of nodes, given in a machine list in a cluster. It must be run having exclusive access of the cluster to reflect the effect of network technology and protocol stack in that cluster. Second, this technique does not take into account the network usage in the cluster, assuming that there is no more applications being executing in the cluster at the same time.

Another studies are focused on specific scenarios in cluster systems [14].This work shows ErrMgr, a recovery policy framework providing applications the ability to compose a set of policies at runtime. This framework is implemented in the Open MPI runtime environment and currently includes three policy options: run-through stabilization, automatic process recovery, and preemptive process migration. The former option supports continuing research into fault tolerant MPI semantics while the latter two provide transparent, checkpoint/restart style recovery. The authors discuss the impact on recovery policy performance for various stable storage strategies including staging, caching, and compression. Results indicate that some stable storage configurations designed to provide low overhead during failure-free execution often negatively impact recovery performance.

3. Basic architecture of the proposed framework

In this paper we propose a framework for reducing the volume of data transmitted through the network by using compression algorithms. Although using compression to increase the transmission speed is not new, our proposal provides several features that alleviate the main drawbacks of the existent methods.

- Dynamic: The proposed framework is able to adapt itself dynamically to the current scenario. This means that it manages different compressing algorithms, such as RLE [15], Huffman [16], Rice [17], FPC [18], and LZO [19], to obtain the best performance depending of the data block to be processed. Moreover,

this approach takes into account the current percentage of network usage before taking a decision about compressing each data block, which adds a high level of flexibility.

- Non-Intrusive: The proposed framework is based on simulation techniques, which does not require exclusive access of the entire cluster system. Basically, our proposed framework only requires executing one benchmark in one node of the cluster to calculate the compression/decompression time of each algorithm. Due to having exclusive access to a cluster is complicated and, in some cases, it is practically impossible, we propose to use a simulated environment to calculate heuristics used to make critical decisions about compressing data.
- Scalable: The proposed framework can be applied to any parallel application, independently of the number of processes and computing nodes used in the cluster system. Moreover, due to the framework uses a simulated cluster, the environments can be scaled up easily by increasing the number of computing nodes. Also, non-existent system may be modelled as well.
- Portable: The proposed framework is not focused on a specific programming language nor a platform. It provides a generic method to calculate when the overall system performance can be achieved by compressing the data sent through the network. Thus, it can be applied to any parallel application that performs inter-node communications.

Figure 1 shows the basic schema of the proposed framework. Initially, a specific benchmark must be executed in a computing node of the cluster (see ① in Figure 1). This benchmark is based on measuring the time required for compressing/decompressing data using different algorithms, such as RLE, Huffman, Rice, FPC, and LZO. Each compression algorithm is executed using different data-sets, where each data-set has a specific features that defines a specific test for compression/decompression. These features consist of the size of the data set to be processed, the data type of this data-set and the redundancy level. The main objective of executing this benchmark is to obtain accurate measurements from a computing node of the cluster, when different compression algorithms are executed using a wide spectrum of configurations. Once this benchmark is executed, a Compression Metrics File is generated. This file contains all the information about compression/decompression time gathered in each test executed in the benchmark.

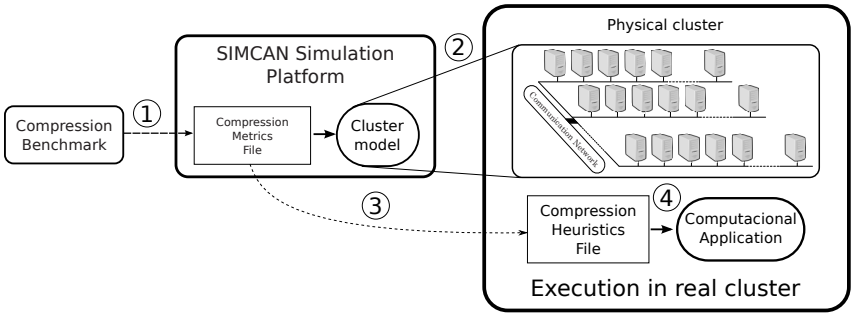

Fig. 1. Global schema of SANComSim.

The next step consists in modelling the cluster (see ② in Figure 1). In this step, the user is responsible of providing both the topology of the cluster and the characteristics of the computing nodes. In order to perform this task, the SIMCAN simulation platform is used. This simulation platform allows user to model any distributed system and parallel and distributed applications. It can be easily achieved since SIMCAN provides a GUI that eases considerably the task of modelling distributed systems. Therefore, computing nodes are modelled by defining 4 basic systems: CPU, memory, network and storage.

Next, SANComSim performs simulations using both the Compression Metrics File and the cluster model generated with SIMCAN (see ③ in Figure 1). The main objective of using simulated environments is that different scenarios can be represented easily in a controlled way. Thus, it is not required the exclusive access to the real cluster, which in most cases is unfeasible due to the continuous activity of several users. Moreover, a wide range of situations can be simulated easily in SIMCAN, such as specific workloads in the communication network, or

Algorithm 1 Generation of Compression Heuristics File

```
1:  load (Compression_Metrics_File);
2:  load (Cluster_Model);
    // Initialization...
3:  networkUsage ← 0;
4:  initProcesses (local, remote);
5:  compressionAlgorithms ← {LRE, HUFF, RICE, LZO};
6:  redundancyLevel ← {0,25, 50, 75, 100};
7:  messageSize ← {100KB, 500KB, 800KB, 1.5MB};
8:  messageType ← {int, float, double, char, random};
    // Generates all possible scenarios in the simulated environment
9:  while (networkUsage ≤ 100) do
10:   setNetworkUsage (networkUsage);
11:   for (each alg ∈ compressionAlgoriths) do
12:     for (each dataType ∈ messageType) do
13:       for (each msgSize ∈ messageSize) do
14:         for (each redLevel ∈ redundancyLevel) do
15:           beforeCompress ← timeStamp();
16:           compressedData ← compress (alg, dataType, msgSize, redLevel);
17:           send (remoteProcess, compressedData);
18:           decompress (alg, dataType, msgSize, redLevel);
19:           afterDecompress ← timeStamp();
              // Calculates compression and decompression time
20:           timeWithCompression ← (afterDecompress - beforeCompress);
21:           beforeSend ← timeStamp();
22:           send (remoteProcess, blockDataType);
23:           afterSend ← timeStamp();
              // Calculates time for sending data without compression
24:           commTime ← (afterSend - beforeSend);
              // Write results in heuristics file
25:           writeInHeuristicsTable (alg, dataType, msgSize, redLevel, timeWithCompression, commTime);
26:         end for
27:       end for
28:     end for
29:   end for
      // Increases network utilization in the cluster
30:   networkUsage ← networkUsage + 10;
31: end while
```

a given number of users using a specific percentage of the computing nodes. Therefore, our framework can be tested using real-case scenarios, which in most cases involves the interaction with different users. The output when these simulations are executed is the Compression Heuristics File. This file contains relevant information about the speed-up obtained for each simulated scenario. Algorithm 1 shows how the Compression Heuristics File is generated.

Finally, the generated Compression Heuristics File must be used for any computational application executed in the real cluster (see ④ in Figure 1). However, this does not imply that for every message sent through the network the compression have to be applied. The proposed framework is dynamically adapted to the current features of the cluster. Algorithm 2 shows how the decision about compressing each message sent is calculated. This algorithm must be included in the application to be executed (see ④ in Figure 1). Initially, values stored in Compression Heuristics File are loaded in internal tables. Then, depending of each message to be sent through the network, the algorithm calculates if applying compression techniques produces an increasing in speed-up. If so, then the

Algorithm 2 send (data, destinationProcess)

```
1: heuristics ← load (Compression_Heuristics_File);
2: netUsage ← getCurrentNetworkBandwidth()
   // Calculates the level of redundancy for current data block
3: redundancy ← calculateRedundancy (data);
   // Selects the best algorithm for compression
4: algorithm ← selectBestAlgorithm (data, data.type, data.size, redundancy)
   // Calculates if using compression improves performance
5: if (getCompressionTime (algorithm, data, netUsage, heuristics) ≥ getCommTime (data, netUsage, heuristics))
   then
6:    sendDataWithoutCompression (data, destination);
7: else
8:    sendCompressed (data, destination, algorithm);
9: end if
```

best compression algorithm is selected and the data is sent compressed to the corresponding destination. On the contrary, if applying any compression algorithm does not increase speed-up, this data is sent uncompressed.

Following, Section 3.1 describes how the compression/de-compression times are calculated, and Section 3.2 presents a brief description of the SIMCAN simulation platform is provided.

3.1. Obtaining compression information

Initially, a thorough analysis of the different compression algorithms must be performed. This analysis consists in executing a synthetic benchmark for calculating the time required for compressing/decompressing data using different algorithms. In our previous work [13], we have found that a message has three main properties that affect the performance of compressing/decompressing data: message size; datatype of each message element; and redundancy level of message data.

Therefore, the benchmark studies the algorithm complexity and the amount of compression for each compressor with different types of data, buffer sizes, and redundancy levels. This benchmark has been evaluated in our cluster with 22 nodes. Each node is Dual-Core Intel(R) Xeon(R) CPU with 4 GB of RAM. The data sets used have been generated synthetically. Each dataset contains buffers with different properties:

- Datatype (integer, floating-point, and double precession).
- Buffer size: The buffer sizes considered are 100 KB, 500 KB, 800 KB, and 1500 KB for integer and floating-point datasets.
- Redundancy levels: 0%, 25%, 50%, 75%, and 100%. If the buffer redundancy level is 0%, it means that all values are different. But, if the redundancy level is 100%, it means that all values are equal to zero. These entries are randomly distributed among the buffer memory map.

3.2. Overview of the SIMCAN simulation platform

SIMCAN is a simulation platform oriented towards analyzing and studying parallel applications on distributed systems [20]. SIMCAN has been designed targeting to provide flexibility, accuracy, performance, and scalability, which makes it a powerful simulation platform for designing, testing and analyzing both actual and non-existent architectures. The range of systems to simulate comes from a single computing node to a complete high performance distributed system. In fact, this simulation platform has been applied to data systems simulation in the EPCC art Edinburgh University [21].

The SIMCAN project was initiated in 2008 and it was released as open source software. It is currently available at "*http://www.arcos.inf.uc3m.es/~SIMCAN*". A thorough validation process of the proposed simulation platform has been fulfilled by comparing the results obtained in real architectures with those obtained in the analogous simulated environments. Those experiments have been achieved in two different real-world scenarios using three different applications: the IOZone benchmark, the mppTest suite, and BIPS3D [20].

SIMCAN provides a GUI to create and manage large simulation scenarios easily and quickly (see Figure 2). This GUI is targeted to manage and generate models for the SIMCAN simulation platform. The main goal of this GUI is two-fold. First, hiding all low-level details, including the language used for implementing components and creating simulated environments. Second, to ease the generation and customization of distributed system models.

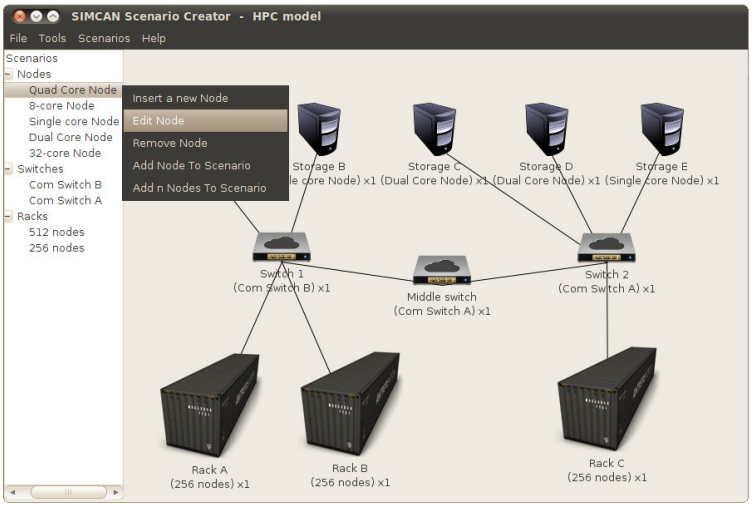

Fig. 2. Screenshot of the SIMCAN's GUI [20]

4. Performance experiments

In this section we present several performance experiments that evaluates the usefulness of SANComSim. The experiments were conducted by executing the BIPS3D application on two different High-Performance Clusters, modelled using the SIMCAN simulation platform. We start with an overview of the BIPS3D application in section 4.1. The BIPS3D application has been modelled using SIMCAN in the past. In these previous cases, the generated models were used to study the performance of BIPS3D in distributed systems using different architectural configurations [22, 23]. Section 4.2 describes the HPC clusters used in our evaluation. The evaluation results themselves are presented in section 4.3.

4.1. Overview of the BIPS3D application

BIPS3D is a 3-dimensional simulation of BJT and HBT bipolar devices [3]. The goal of the 3D simulation is to relate electrical characteristics of the device with its physical and geometrical parameters. The basic equations to be solved are Poisson's equation and electron and hole continuity, in a stationary state. Finite element methods are applied in order to discretize the Poisson equation, hole and electron continuity equations by using tetrahedral elements. The result is an unstructured mesh.

Using the METIS library [24], the meshes are divided into sub-domains, in such a manner that one sub-domain corresponds to one process. The next step is decoupling the Poisson equation from the hole and electron continuity equations. They are linearized using the Newton method. Then we construct the part corresponding to the associated linear system for each sub-domain in a parallel manner. Each system is solved using domain decomposition methods. Finally, the results are written to a file.

This simulator has been written in C and Fortran. Also, the MPI library has been used to communicate processes through message passing. Thus, BIPS3D can be used in a multi-computer environment, parallelizing computing and improving the performance.

4.2. Overview of the HPC systems modelled

The experiments performed have been executed in a simulated environment using the SIMCAN simulation platform. The configuration of each modelled HPC cluster is following described:

1. Cluster 1: This cluster consists of 256 nodes. Each node is Dual-Core Intel(R) Xeon(R) CPU with 4 GB of RAM. The network used is a Ethernet Gigabit.
2. Cluster 2: This cluster consists of 512 nodes. Each node is Intel Pentium 4 CPU 3.20GHz with hyper-threading and 4 GB of RAM, Cache size of 1024 KB. The network used is a Ethernet Gigabit.

4.3. Evaluation

In this section we analyze the performance obtained by using SANComSim in two different clusters. Each cluster has been modelled using SIMCAN. Moreover, Algorithm 2 has been included in the modelled application BIPS3D. Thus, we prove the portability of our proposed framework by using it in two different HPC modelled environments. However, our framework is not limited to simulation environments, it can be also used with real applications in a hardware-based clusters.

Figure 3 and Figure 4 show the overall speed-up obtained by using SANComSim. Each experiment has been executed with 4, 8, 16, 32, 64 and 128 processes. Moreover, the percentage of network usage in each cluster has been configured in slices of 10%, starting from 0 (exclusive use of the cluster) and finishing in 100% (saturated network). The speed-up in these charts is calculated using the following formula:

$$speed - up = \frac{Time_{BIPS3D}}{Time_{BIPS3D\_SANComSim}}$$

where $Time_{BIPS3D}$ is the time required to execute BIPS3D in SIMCAN without applying any compression technique, and $Time_{BIPS3D\_SANComSim}$ is the time required to execute BIPS3D in SIMCAN using the SANComSim framework.

In most cases, the speedups are achieved when applying our proposed framework. This improvement in the overall system performance is due to applying compression techniques reduces considerably the amount of data to be transferred between processes, which reduces the total execution time as well. However, in some cases, SANComSim does not use compression because the pre-calculated heuristics show that it results in a

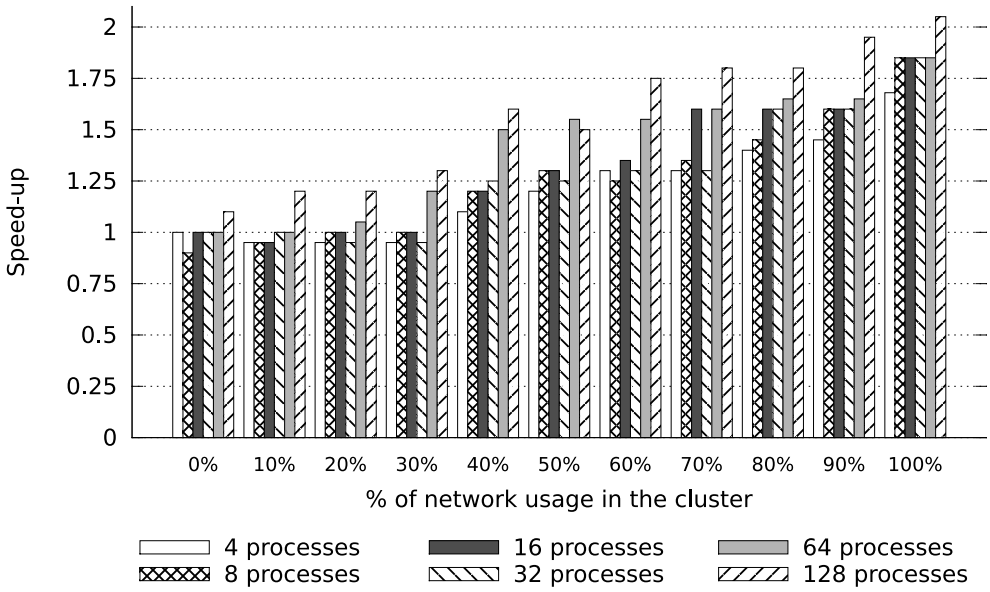

Fig. 3. Speed-up of executing BIPS3D using SANSComSim in Cluster 1

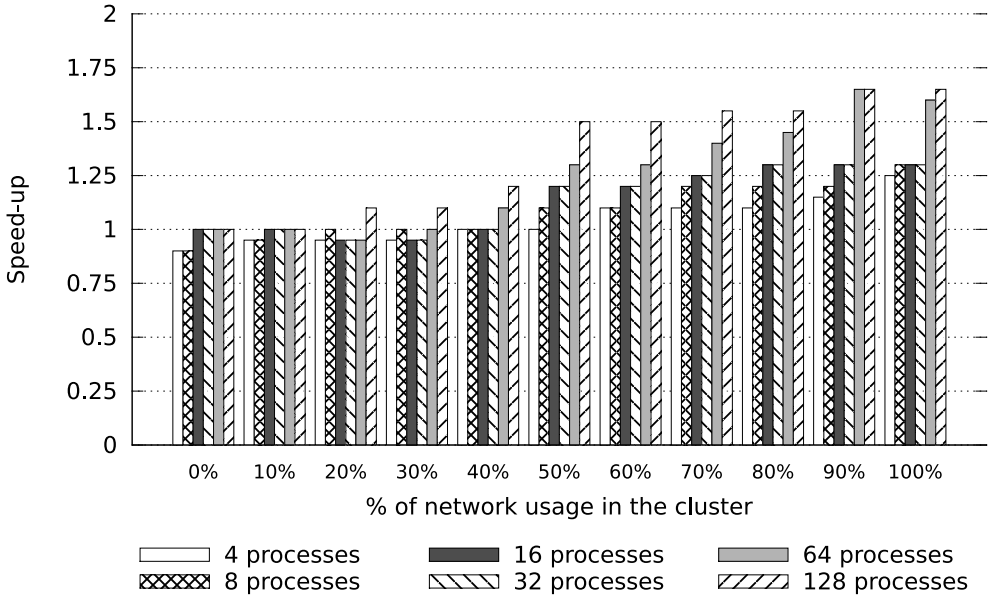

Fig. 4. Speed-up of executing BIPS3D using SANSComSim in Cluster 2

worst performance. These cases are shown when the speed-up is equal to 1. There are very few cases when the performance obtained is less than 1, which means that executing without applying SANComSim would obtain a better performance. The main reason of this small drop of performance is two-fold: First, the pre-calculated heuristic of a specific message is not as accurate as it would be, which force the model to apply compression when it would not be applied. Second, the difference of time between applying compression is practically insignificant. Thus, the time required for calculating the level of redundancy exceeds this difference of time. Consequently, the final performance obtained is worst than sending this message without applying compression.

In general, both charts show that when the number of processes increases, the speed-up increases as well. The same goes with the percentage of network usage. The better speed-up results are obtained when there is activity in the network, being the best scenario when the network is saturated (100% network usage). This is caused because when compression is not used, the volume of data sent through the network is greater, and consequently it requires more time to be transferred. This time grows when the percentage of network usage is increased as well.

The difference between the speedups achieved in the two scenarios is due to the cluster architecture. Although the network used in both clusters is the same, the processors (i.e. Cluster 2 contains hyper-threading cores) and the number of computing nodes are different. In the Cluster 2 architecture, the SANComSim is deactivated more often (see speed-up values equal to 1 in Figure 4).

5. Conclusions and Future work

In this paper we have presented a framework for optimizing the network performance in computational applications, called SANComSim. The main goal of this framework is to provide a portable, scalable, adaptive and non-intrusive method for reducing the volume of data transferred through the network by using compression techniques. Moreover, in this paper has been proven its level of flexibility because SANComSim have been used in two different simulated HPC environments.

The evaluation results show that SANComSim provides an increasing in the speed-up of BIPS3D for most of the scenarios considered. In those scenarios where the speed-up obtained is practically 1, the compression is not activated and then data is sent uncompressed through the network. There are very few cases where the performance obtained is below 1. This is mainly caused by the heuristics. In some cases, the overhead of calculating the level

of redundancy is greater than the speed-up obtained in the compression Thus, there is a practically insignificant loss in the overall system performance. In general, the overall performance gain increases when the number of processes increases as well, which demonstrates the scalability of our proposed framework.

As future work we would evaluate more compression algorithms using a wider spectrum of parameters with the purpose to obtain more accurate heuristics, like message size. Also, more applications and HPC systems would be tested with our proposed framework.

Acknowledgements

This research was partially supported by the Spanish MEC projects TESIS (TIN2009-14312-C02-01) and ESTuDIo (TIN2012-36812-C02-01).

References

[1] T. M. Mitchell, R. Hutchinson, R. S. Niculescu, F. Pereira, X. Wang, M. Just, S. Newman, Learning to decode cognitive states from brain images, Machine Learning 57 (2004) 145–175.
[2] W. Michael Brown and Trung D. Nguyen and Miguel Fuentes-Cabrera and Jason D. Fowlkes and Philip D. Rack and Mark Berger and Arthur S. Bland, An Evaluation of Molecular Dynamics Performance on the Hybrid Cray XK6 Supercomputer, Procedia Computer Science, Proceedings of the International Conference on Computational Science (ICCS'12) 9 (0) (2012) 186–195.
[3] A. Loureiro, J. González, T. Pena, A parallel 3D semiconductor device simulator for gradual heterojunction bipolar transistors, Journal of Numerical Modelling: electronic networks, devices and fields 16 (2003) 53–66.
[4] Y. Jia, P. Luszczek, J. Dongarra, Multi-GPU Implementation of LU Factorization, Procedia Computer Science, Proceedings of the International Conference on Computational Science (ICCS'12) 9 (0) (2012) 106–115.
[5] H. Meuer, E. Strohmaier, J. Dongarra, H. D. Simon, Top500 Supercomputer sites, http://www.top500.org (2012).
[6] D. A. Lelewer, D. S. Hirschberg, Data compression, ACM Comput. Surv. 19 (3) (1987) 261–296.
[7] J. Carretero, J. No, P. Chen, COMPASSION: a Parallel I/O Runtime System Including Chunking and Compression for Irregular Applications, in: Solving Irregularly Structured Problems in Parallel, LNCS vol. 1457 , Springer Berlin Heidelberg, 1998, pp. 262–273.
[8] D. Balkanski, M. Trams, W. Rehm, Heterogeneous Computing with MPICH/Madeleine and PACX-MPI: a Critical Comparison, Technische Universitat Chemnitz (2003).
[9] R. Keller, Using pacx-mpi in metacomputing applications, in: 18th Symposium Simulations technique, Erlangen, 2005.
[10] P. Ratanaworabhan, J. Ke, M. Burtscher, Fast Lossless Compression of Scientific Floating-Point Data, in: Proceedings of the Data Compression Conference (DCC '06), IEEE Computer Society, Washington, DC, USA, 2006, pp. 133–142.
[11] J. Ke, M. Burtscher, E. Speight, Runtime Compression of MPI Messages to Improve the Performance and Scalability of Parallel Applications, in: Proceedings of the 2004 ACM/IEEE conference on Supercomputing (SC '04), IEEE Computer Society, Washington, DC, USA, 2004, p. 59.
[12] R. Filgueira, D. Singh, A. Calderon, J. Carretero, CoMPI: Enhancing MPI Based Applications Performance and Scalability Using Run-Time Compression, in: Recent Advances in Parallel Virtual Machine and Message Passing Interface, LNCS vol. 5759, 2009, pp. 207–218.
[13] R. Filgueira, J. Carretero, D. E. Singh, A. Calderón, A. Núñez, Dynamic-coMPI: Dynamic optimization techniques for MPI parallel applications, The Journal of Supercomputing 59 (1) (2012) 361–391.
[14] J. Hursey, A. Lumsdaine, Technical Report TR686: A Composable Runtime Recovery Policy Framework Supporting Resilient HPC Applications, Tech. rep., Indiana University, School of Informatics and Computing (2010).
[15] R. Zigon, Run length encoding, Dr. Dobbs Journal of Software Tools 14 (2) (1989) 126–128.
[16] D. E. Knuth, Dynamic Huffman coding, J. Algorithms 6 (2) (1985) 163–180.
[17] D. S. Coco, V. DArrigo, A Rice-based Lossless Data Compression System For Space, in: Proceedings of the 2000 IEEE Nordic Signal Processing Symposium, 2000, pp. 133–142.
[18] M. Burtscher, P. Ratanaworabhan, FPC: A High-Speed Compressor for Double-Precision Floating-Point Data, IEEE Trans. Comput. 58 (1) (2009) 18–31.
[19] M. F. Oberhumer, LZO real-time data compression library, http://www.oberhumer.com/opensource/lzo (2013).
[20] A. Núñez, J. Fernández, J. D. García, F. García, J. Carretero, SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications, Simulation Modelling Practice and Theory 20 (1) (2012) 12–32.
[21] J. Fernández, L. Han, A. Núñez, J. Carretero, J. van Hemert, Using architectural simulation models to aid the design of data intensive application, in: 3rd International Conference on Advanced Engineering Computing and Applications in Sciences, ADVCOMP'09, IEEE Press, 2009, pp. 163–168.
[22] A. Núñez, J. Fernández, J. D. García, F. García, J. Carretero, New techniques for simulating high performance MPI applications on large storage networks, The Journal of Supercomputing 51 (1) (2010) 40–57.
[23] A. Núñez, J. Fernández, J. D. García, J. Carretero, Analyzing Scalable High-Performance I/O Architectures, in: PDPTA'08, The 2008 International Conference on Parallel and Distributed Processing Techniques and Applications, 2008, pp. 631–637.
[24] G. Karypis, V. Kumar, METIS A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices., Tech. rep., Department of Computer Science/Army HPC Research Center, University of Minnesota, Minneapolis (1998).