Universidade Federal do Rio Grande do Sul
Instituto de Informática
Grupo de Processamento Paralelo e Distribuído

XIII Workshop de Processamento Paralelo e Distribuído
WSPPD 2015
(http://inf.ufrgs.br/gppd/wsppd/2015)

21 de Agosto de 2015
Auditório do Centro de Eventos do Instituto de Informática
Porto Alegre, RS

Proceedings

(ISSN: 2175-6848)

GPPD

INSTITUTO DE INFORMÁTICA UFRGS

Analyzing Energy Consumption of Elastic HPC Applications in the Cloud

Gustavo Rostirolla, Vinicius Facco Rodrigues, Rodrigo da Rosa Righi, Cristiano André da Costa
Universidade do Vale do Rio dos Sinos, Applied Computing Graduate Program
grostirolla1@gmail.com, viniciusfacco@live.com, {rrrighi,cac}@unisinos.br

Abstract

One of the main characteristics of cloud computing is elasticity, which refers to the capacity of on-the-fly changing the number of resources to support the execution of an application, or service. One of the main challenges in this scope is how to measure its effectiveness, because of elasticity enables high performance computing by reducing the application time, but we can pay an inviable amount of resource and/or energy to accomplish this. Particularly, we observed that the state-of-the-art does not present an energy consumption model that fits a malleable number of resources, but only a fixed and predefined number of them. In this context, we are proposing in this article E2M (Elastic Energy Model) an elastic energy consumption model, which analyses traces from the CPU and main memory to estimate the total energy to execute a HPC application in the cloud. The results revealed a median accuracy of 97.15% when analyzing data from the model against real energy consumption data.

1. Introduction

One of the key features of cloud computing is elasticity, in which users can scale their computational resources up or down at any moment, according to the demand or the desired response time [4]. Considering a very long running parallel application, an user may want to increase the number of instances to try to reduce the completion time of the application. Logically, the success of this procedure will depend on both the computational grain and application modeling. On the other hand, if an application is not scaling in a linear or close to linear fashion, and if the user is flexible with respect to the completion time, the number of instances can be reduced. This results in a lower nodes $\times$ hours index, and thus in a lower cost and better energy saving.

Elasticity can be a prejudicial when considering performance and energy consumption. Directly related to both we have the resource consumption, that can also help in measuring the elasticity quality. Although elasticity allows applications to acquire and release resources dynamically, adjusting to changing demands, deciding about appropriate thresholds and measuring performance and energy consumption accurately are not easy tasks [4].

In this way, this article presents E2M (Elastic Energy Model), an elastic energy consumption model which gives data about energy when executing HPC applications in elastic-based cloud environments. Particularly, E2M extracts energy consumption data from a malleable infrastructure of resources, enabling relationships among energy consumption, resource consumption and performance. Aiming at analyzing the energy model, we are using a previous work named AutoElastic [7], which is a reactive-driven elasticity middleware that manages cloud resources according to the demand of a HPC application. Thus, E2M acts as an AutoElastic plugin, saving energy data during the application runtime.

The remainder of this article will first introduce E2M in Section 2. The evaluation methodology and the discussion of the results are described in Section 3. Section 4 introduce the related work. Finally, Section 5 presents final remarks, highlighting the contributions with quantitative data and presenting directions on future work.

2. E2M - Elastic Energy Model

Deploying energy sensors or wattmeters can be costly if not done at the time the whole infrastructure (*i.e.*, cluster or data center) is set up, besides being time consuming as the infrastructure scales up. An alternative and less expensive solution is to use energy models to estimate the consumption of components or of an entire infrastructure [6]. We are following the same methodology used by Luo et al. [5] to develop power model which consists of three phases: (i) Collect samples of resource usage, and the machine energy consumption using a smart power meter. In our case, we used a Minipa power meter model ET-4090 and collected data from more than 8 thousand samples using a composite load that may consume multiple types of cloud resources in order to represent real cloud applications [1]; (ii) Perform regression methods to generate the energy model to be used later; (iii) Test the model in a different set of data collected

from another homogeneous machines in order to validate if the model works properly among the machines.

In order to analyze the model precision we collected CPU, main memory and instantaneous power consumption data and applied Principal Component Regression (PCR) over more than 8 thousand samples obtained from a single node. The collected data are in line with previous studies [6], that present the CPU as the main responsible for the nodeś energy consumption. After that, we predicted the same amount of power samples using CPU and memory data collected from another node with same hardware configuration. Comparing this power samples with the samples collected during the whole execution with the power meter we obtained a mean and median accuracy of 97.15% and 97.72%, respectively.

After the application execution we input the CPU and main memory in the trained model in order to obtain an instantaneous power consumption measured in Watts (W). The great advantage of E2M is the fact that it considers the cloud elasticity, in other words the model takes into account only the power consumption of resources that were actually being used, not the total consumption of the datacenter, or a specific node.

To complement this analysis we present a set of equations that allow the calculation of elastic energy consumption and also the amount of energy spent by a determined number of nodes. Equation 2 results in the energy consumption of machine m according to the logged CPU and memory value of a machine in a instant i using Equation 1 as basis. Equation 3 is used to calculate the total power consumption of all machines allocated in an instant t, i.e. taking into account elasticity, returning the consumption in Watts. Equation 4 calculates the total energy consumption from an instant 0 to an instant t where the time intervals are computed in seconds and uses the aforementioned Equation 3 that already considers the cloud elasticity, this calculus results in the energy consumption in Joules ($W \times second$). Finally, Equation 5 presents the application power consumption when employing an specific amount of nodes represented by z. This calculus results in the total energy consumption also represented in Joules spent by this amount of nodes.

$$f(CPU, Memory) = \alpha + \beta \times CPU + \delta \times Memory \quad (1)$$

$$MC(m, i) = f(CPU(m, i), Memory(m, i)) \quad (2)$$

$$ETC(t) = \sum_{i=0}^{Machines} MC(i,t) \times x \begin{cases} x = 0 & \text{if machine } i \text{ isn't act. in } t; \\ x = 1 & \text{if machine } i \text{ is act. in } t. \end{cases} \quad (3)$$

$$TC(t) = \sum_{i=0}^{t} ETC(i) \begin{cases} 0 \leq t \leq TotalApplicationTime \end{cases} \quad (4)$$

$$NEC(z) = \sum_{i=0}^{AppTime} ETC(i) \times y \begin{cases} y = 0 & \text{if in } i \text{ the tot. act. mach. } \neq z; \\ y = 1 & \text{if in } i \text{ the tot. act. mach. } = z. \end{cases} \quad (5)$$

3. Results Analysis

Our experiments were conducted using an OpenNebula private Cloud with 6 homogeneous nodes. We are using 2.9 GHz dual-core nodes with 4 GB of RAM memory and an interconnection network of 100 Mbps. A total of four load patterns (Ascending, Constant, Descending and Wave), illustrated in Figure 1, were used along with AutoElastic [7] with and without enabling the elasticity feature. In the case where the elasticity is enabled, the thresholds used were 70% and 90% for the upper threshold and 30% and 50% for the lower threshold.

Figure 2 illustrates the power consumption in Watts using E2M when elasticity actions are disabled. In this context, a single node with two VMs is being used to host slave processes. Here, we observe that the simple fact of turning on the compute node (running Ubuntu Linux Operating System and AutoElastic software) spends about 40 Watts. Any computation activity causes an elevation of this index to the interval (40-71). Although the Ascending function grows slowly, the power consumption here increases quickly up to the upper bound of the interval. The same behavior appears in the Descending and Wave functions.

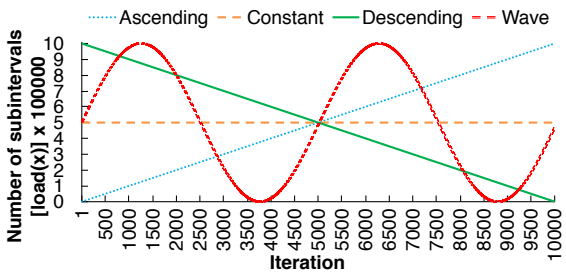


Figure 1. Load patterns used among with AutoElastic.

Figure 3 presents an execution graph highlighting peaks and sudden drops of power consumption when analyzing the Descending function for the four combinations of thresholds. In part (i), we have a host allocation (and 2 VMs inside it), which results in a sudden drop followed by energy consumption increasing thereafter. In addition, we can observe oscillations during the bootstrapping of the VMs, as seen in part (ii). After this procedure, the AutoElastic manager notify the master process that there are new resources

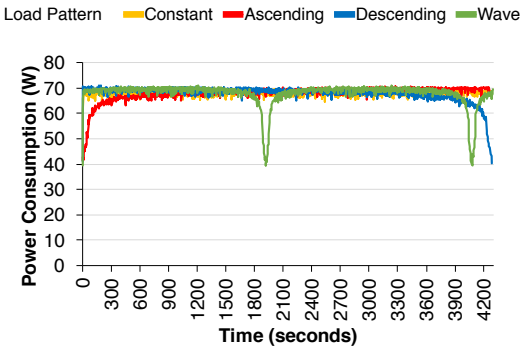


Figure 2. Power consumption behavior of different load patterns without elasticity.

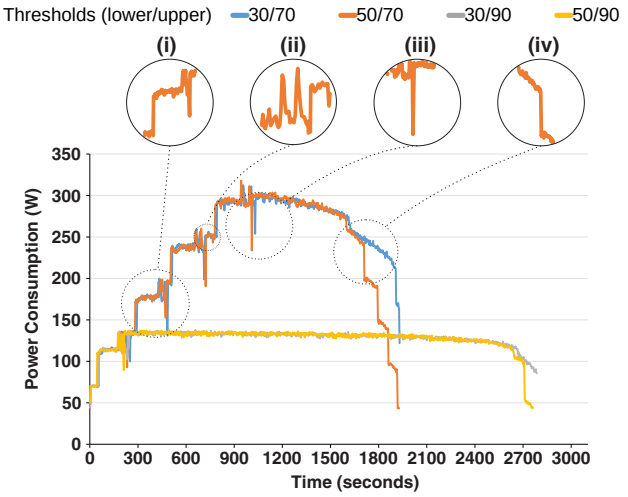


Figure 3. Power consumption behavior of Descending load patterns with varying thresholds. (i) Host allocation, (ii) Virtual machines booting, (iii) Processing stop to incorporate new resources, and (iv) Host Deallocation.

to be used. Part (iii) explores the situation, in which the master reorganizes the communication topology, causing an abrupt decline in power consumption because no CPU usage occurs. The application proceeds in a decreasing curve of load; therefore, resources are deallocated as illustrated in part (iv). Here, we have the shutdown of the VMs and the node is subsequently consolidated.

Considering a non elastic energy model the instantaneous energy consumption lower bound would be $200W$ since each machine IDLE power consumption is $40W$ highlighted by the α value in Equation 1. Figure 4 presents the rest of the execution graphs highlighting peaks and sudden drops of power consumption when analyzing the power consumption in a elastic way using Equation 3 during the total application time. In this graphs we can observe hosts allocation and deallocation, oscillations during the VMs bootstrapping. This graphs presents the advantages in analyzing the application using E2M, since it considers only the power consumption of machines performing computation.

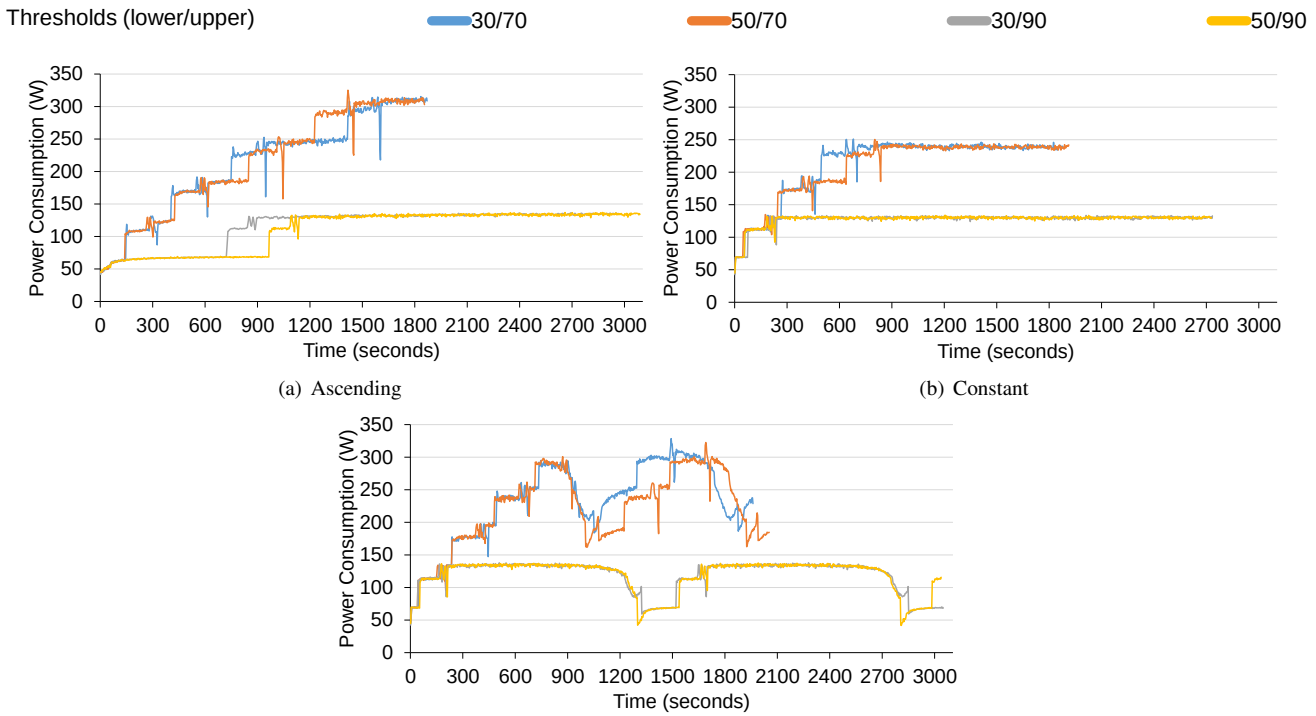
4. Related Work

Some works are focusing in energy models to estimate the energy consumption in cloud environments. However, these works do not focus in the elasticity behavior of such systems. Luo et al. [5] present an energy-aware resource management algorithm need to consider both energy consumption and QoS (Quality of Services) requirements. The article presents a model to predict energy consumption inside a single machine, besides a simulation-driven framework to evaluate energy-aware resource schedule algorithms. Garg et al. [3] present a data center energy model based in the CPU parameter.

Considering the analysis of energy consumption, some works focus in define energy profiles [2], cost evaluation [8] and energy performance [8]. Regarding energy consumption, the traditional method that takes into account power and time is normally employed. In this way, we highlight the following regarding the evaluation metrics: (i) energy consumption evaluation considering a malleable number of resources; (ii) in elastic environments, there is a lack of joint-analysis of energy consumption and resource usage in the context of HPC applications.

5. Conclusion

This article focused on presenting and evaluating an elastic energy consumption model named E2M for cloud data centers. E2M estimates energy consumption based on CPU and Memory traces with mean and median accuracy of 97.15% and 97.72% respectively. It was employed together with the AutoElastic middleware, which runs HPC applications, allocating and deallocating resources according to the processes' demands. The results showed the best energy saving values when using an upper threshold close to 90%, and the worst values for this metric when employing 70%. However, this last case refers to the best performance rates. Focusing on reproducibility, we introduced a set of equations that allows other researchers to employ E2M to measure energy consumption on their elastic HPC applications in a proper way. Finally, we intend to continue our research in order to extend E2M to include heterogeneous machines,



(a) Ascending

(b) Constant

(c) Wave

Figure 4. Power consumption behavior of different load patterns varying thresholds.

since the current version assumes only computational nodes and virtual machines with the same configuration, and IoT middlewares power consumption.

References

[1] F. Chen, J. Grundy, J.-G. Schneider, Y. Yang, and Q. He. Automated analysis of performance and energy consumption for cloud applications. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 39–50, New York, NY, USA, 2014. ACM.

[2] F. Chen, J. Grundy, J.-G. Schneider, Y. Yang, and Q. He. Automated analysis of performance and energy consumption for cloud applications. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 39–50, New York, NY, USA, 2014. ACM.

[3] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya. Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *J. Parallel Distrib. Comput.*, 71(6):732–749, June 2011.

[4] T. Lorido-Botran, J. Miguel-Alonso, and J. Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592, 2014.

[5] L. Luo, W. Wu, W. Tsai, D. Di, and F. Zhang. Simulation of power consumption of cloud data centers. *Simulation Modelling Practice and Theory*, 39(0):152 – 171, 2013. S.I.Energy efficiency in grids and clouds.

[6] A.-C. Orgerie, M. D. D. Assuncao, and L. Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Computing Surveys*, 46(4):1–31, Mar. 2014.

[7] R. Righi, V. Rodrigues, C. Andre daCosta, G. Galante, L. Bona, and T. Ferreto. Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *Cloud Computing, IEEE Transactions on*, PP(99):1–1, 2015.

[8] S. Tesfatsion, E. Wadbro, and J. Tordsson. A combined frequency scaling and application elasticity approach for energy-efficient cloud computing. *Sustainable Computing: Informatics and Systems*, 4(4):205 – 214, 2014. Special Issue on Energy Aware Resource Management and Scheduling (EARMS).

eGPU for Monitoring Performance and Power Consumption on Multi-GPUs

John A. G. Henao[1,2], Víctor M. Abaunza[2], Philippe O. A. Navaux[2], Carlos J. B. Hernández[1]

[1]High Performance and Scientific Computing Center, University Industrial of Santander, BGA-Colombia
[2]Parallel and Distributed Processing Group, Informatics Institute, Federal University of Rio Grande do Sul, POA-Brazil
[1,2]john.garcia1@correo.uis.edu.co, [2]victor.martinez@inf.ufrgs.br, [2]navaux@inf.ufrgs.br, [1]cbarrios@uis.edu.co

Abstract

The evaluation of performance and power consumption is a key step in the design of applications for large computing systems, such as supercomputers, clusters with nodes that have manycores and multi-GPUs. Researchers must design several experiments for workload characterization by observing the architectural implications of different combinations of parameters, such as problem size, number of cores per GPUs, number of process MPI, and observe the resulting clock frequency, memory usage, bandwidth, and power consumption, factors which determine the performance and energy efficiency of their workload implementation. The major problem in performance evaluation consists on the design space exploration given so many parameters. If the resulting data is not properly collected and the parameters of evaluation performance are not well organized, the design exploration may lead to wrong conclusions. This paper presents eGPU such as monitoring tool that can be used on evaluations of performance with many experiments which aim measure and understand the behavior of factors that determine the energy efficiency in nodes with multi-GPUS. eGPU is a monitor to centralize and automate the data captured in runtime while is executed in parallel with the Linpack Benchmark and displays information via sequenceplots, statistical tables, bar graph and shows results in terms of energy efficiency.

1. Introduction

The power consumption is becoming the major concern for Exascale Systems. *"Indeed, a key observation from the study from DARPA (Technology Challenges in Achieving Exascale Systems) is that it may be easier to solve the power problem associated with base computation than it will be to reduce the problem of transporting data from one site to another on the same chip, between closely coupled chips in a common package, or between different racks on opposite sides of a large machine room, or on storing data in the aggregate memory hierarchy"* [1]. In this order, the evaluation of performance and power measurement need to analyze multiple tests under different combinations of parameters to observe the key factors that determine the energy efficiency in terms of energy per computation, energy per data transport, energy per memory access, or energy per secondary storage unit of a workload model in computing system extrapolation of science problems of today to Exascale.

The workload widely used by the Green500 and the Top500 in evaluation of performance and power measurement evaluation of supercomputers is the LINPACK Benchmark (HPL) [2]. The Linpack is an algorithm to solve a dense system of linear equations, where is allowed increase the problem size to find the performance numbers that reflect the largest problem can be run on a supercomputer [3]. The goal of this paper is to present the utilization of the first version of eGPU for automate the data collection that includes clock frequency, memory usage, and power consumption across the sensors of the NVIDIA GPU.

2. eGPU Monitor Structure

eGPU is a type of batch monitor that is formed by two levels [4]. One level to capture data in runtime, composed by three events, and a separate level that can be later used to visualize data online, composed by four events, such as shown in the Figure 1.

The first level is used to run a script called $eGPUrecord.sh$, where the execution steps and exported libraries needed to run Linpack Benchmark are declared, such as BLAS Library and many oth-

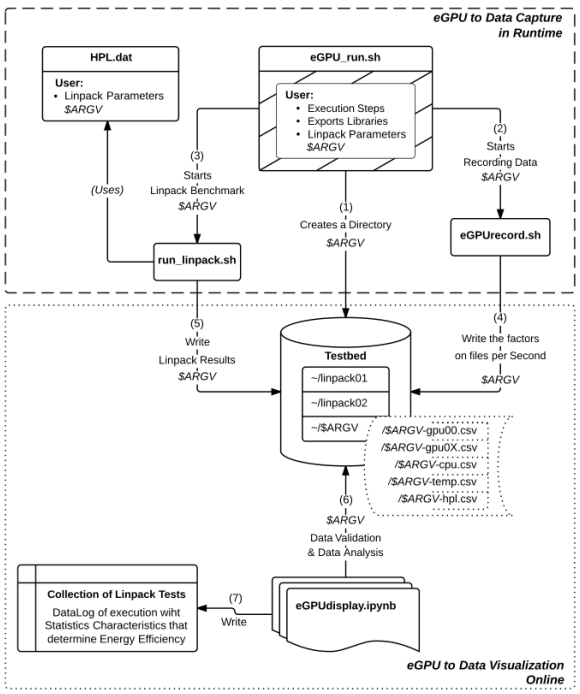


Figure 1. eGPU Monitor Structure.

ers: $(-lcuda - lcudart - lcublas - lmkl\_intel\_lp64 - lmkl\_intel\_thread - lmkl\_core)$. The user writes the same Linpack parameters defined in the file $HPL.dat$ to execute the experiment. Those parameters are used as a unique key called $ARGV$ to share the argument in all the following events that uniquely identify each test of Linpack benchmark. The event (1) creates a directory in a repository that we called $testbet$, where the data is centralized. The events (2) and (4) are used to run a script called $eGPUrecord.sh$ with special tasks written in AWK, C and environment variables, for text processing and data collection in the $testbet$ repository. $eGPUrecord.sh$ makes the data extraction and writes: one separate file for each GPU with the data (Streaming Multiprocessor Clock Frequency -SM-, Memory Clock Frequency, Memory Usage, Power Consumption), these all have new data registered every second. The events (3) and (5) start the script $run\_linpack$ to solve a dense system of linear equations, calculate performance obtained and write results when it finishes.

The second level is used at post-processing for data visualization. It displays information via sequence plots, statistical tables, histograms and shows results in terms of energy efficiency. The event (6) displays information of tests through the IPython Notebook Viewer in a program called $eGPUdisplay.ipynb$ that contains predefined code routines written in Python, where the researchers have to type the argument $ARGV$ to identify the unique test of the Linpack benchmark. eGPU will then shows the sequence plots for the characteristics of each GPU in each node and the following functions. At this point, it will also make a data validation with arguments received from the events (2) and (3) to shows statistical tables with the mean and standard deviation for all the data of each GPU of each node. Subsequently eGPU displays the Linpack results with time spent, performance, power consumption and energy efficiency. Finally eGPU displays a bar graph with a comparison between the node power consumption when idle and when running the Linpack Benchmark.

3. Experimental Procedures and Results

This section presents the use of eGPU to capture data and visualize it in one test with Linpack benchmark. The data were captured in the evaluation of the influence of the number of GPUs and processes in the performance and energy efficiency on nodes of the GUANE cluster.[1] The computational resources used for this experiment are part of one node of the 'A' settings of cluster GUANE that consists in ProLiant SL390s-G7 computing nodes such as shown in Table 1.

Setting GUANE	A	B	C
Node type	SL390s	SL390s	SL390s
Number of nodes	8	3	5
Processor Intel	Xeon	Xeon	Xeon
Processor Model	E5645	E5645	E5640
Processor by node (#)	2	2	2
Clock frequency (GHz)	2.40	2.40	2.670
Core/Processor (#)	6	6	4
Thread/Core (#)	2	2	2
GPUS Nvidia	Tesla	Tesla	Tesla
GPUS Model	M2075	M2050	M2050
GPUs by node (#)	8	8	8
Memory DDR (GB)	104	104	104
SAS disk (GB)	200	200	200
Gigabit Ethernet (Gbps)	10	10	10
InfiniBand (IB)	1	1	1

Table 1. Settings GUANE Cluster Nodes.

1 GUANE is a heterogeneous cluster in the High Performance and Scientific Computing Center -SC3UIS, located at technological camp of Industrial University of Santander at Piedecuesta, a municipality of the metropolitan area of Bucaramanga in Santander, Colombia.

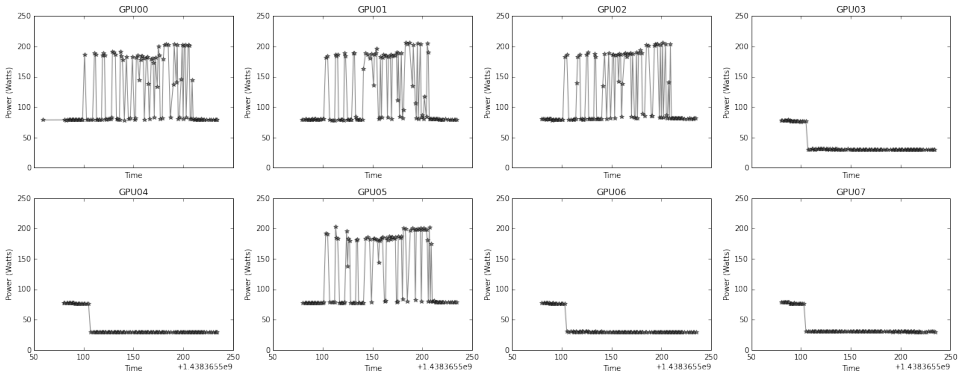


Figure 2. eGPU-Sequenceplot to analyzes of Power Consumption by each GPU.

In this experimental procedure we used HPL.2.0 version configured for Tesla GPUs [5]. The Linpack parameters used are shown in Table 2. They were written in the file $HPL.dat$ and the script $eGPU\_run.sh$ to perform the test for the Linpack benchmark. Then eGPU starts to do text processing and centralize the data collection in the repository $testbed$, and links all events of $\$ARGV$, which represents a unique structure with the parameters used for this test. The script $run\_linpack$ writes performance results in the repository $testbet$ when it is finished.

Matrix size	49152
Block size	1024
GPU Used	4
Cores per GPU	3
Process MPI	4

Table 2. The Linpack parameters used in experimental procedure.

After having captured data, an iterative session is created via $tunneling\ ssh$ to execute a remote IPython Notebook Viewer of the second level of eGPU, in order to visualize data. Once the program $eGPUdisplay.ipynb$ is initialized, we must provide the path with the argument $\$ARGV$ for this experimental procedure. Below we show eight sequence plots with the data (SM Clock Frequency, Memory Clock Frequency, Memory Usage, Power Draw) for each GPU of one node. In the Figure 2, we show the sequence plots for Power Consumption of each GPU, enabling us to observe which GPUs worked while Linpack executed. Next, eGPU shows the statistics to analyze the data that determines energy efficiency such as shown in Table 3. This experiment procedure shows the 4 GPUs that are inactive down to average frequencies of 212MHz for SM clock and 340MHz for memory clock, in order to reduce power consumption. The other 4 GPUs actually working have constant frequencies of 1147MHz for SM clock and 1566MHz for memory clock, with a constant memory usage of 2128MiB, thus generating an average power of 120 watts between working GPUs. The standard deviation of 51 watts shows that execution can be improved, for instance, if we increase the number of cores per GPU or the number of MPI processes to make full utilization of GPUs cores.

Subsequently eGPU displays the amount of energy used by each GPU between $idlepower$ and $runpower$, in terms of Joules. Idle power is the power consumption when a GPU is on but no application is running, for which we found an average power consumption of 78.82 Watts for all GPUs in the node. In comparison, when the GPUs are running, the average power consumption is 78.7 Watts, since the 4 GPUs that are inactive scale down the SM and memory frequencies, such as shown in Figure 3.

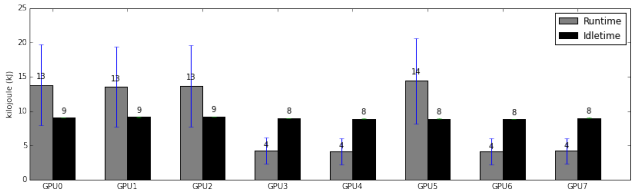


Figure 3. eGPU-Bar graph to analysis of energy used between Idletime and Runtime by each GPU.

Factors	GPU00	GPU01	GPU02	GPU03	GPU04	GPU05	GPU06	GPU007
SM(MHz) Mean	1147	1147	1147	215.74	223.73	1147	204.66	206.65
SM(MHz) Std	0	0	0	282.46	301.54	0	262.64	268.84
Memory(MHz) Mean	1566	1566	1566	346.53	346.53	1566	334.09	334.09
Memory(MHz) Std	0	0	0	507.90	507.90	0	495.24	495.24
MEM Usage(MiB) Mean	2128.9	2128.89	2128.9	10	10	2128.87	10	10
1566 MEM Usage(MiB) Std	0.30	0.31	0.30	0	0	0.33	0	0
Power(Watts) Mean	120.55	118.01	119.12	37.22	36.27	125.65	36.07	36.76
Power(Watts) Std	51.29	50.92	51.36	16.62	16.78	54.32	16.26	16.00
Energy(KJ) Mean	13.80	13.51	13.64	4.26	4.15	14.39	4.13	4.21
Energy(KJ) Std	5.87	5.83	5.88	1.90	1.92	6.22	1.86	1.83

Table 3. eGPU-Statistics to analyzes the factors that determine Energy Efficient.

Table 4 shows the results obtained for performance, where eGPU observed 691.20 GFLOPS and time spent solving the Linpack, and calculated a latency between the time you start to capture data and the time eGPU start Linpack. When using 4 GPUs, with 3 cores per GPU and 4 processes MPI to solve a 49152∗49152 system of linear equations, one node of Cluster GUANE achieved the energy efficiency of 1097.68 MFLOPS/W.

eGPUrecord Time:	121 sec
eGPUrecord Latency:	6 sec
HPL2.0 Time:	114.54 sec
HPL2.0 Performance:	691.20 GFLOPS
Power Consumption:	629.65 Watts
Energy Consumption:	72.12 kJ
Energy Efficiency:	1097.68 MFLOPS/W

Table 4. eGPU-Results to analyzes of Linpack benchmark in this experimental procedure.

4. Conclusions

We constructed eGPU to facilitate the collection and visualization of data to analyze many tests under different combinations of parameters and observe the granularity of the factors that determine energy efficiency in clusters with multi-GPUs. The method we use is focused ony analyzing previously compiled applications, where researchers do not need to orchestrate the code to execute eGPU, ensuring the integrity of the results. Based on the experiment procedures and results presented, eGPU is a good alternative to analyze power consumption in clusters with multi-GPUs from a software level, and can be complemented with other energy monitors that are designed to be plugged-in directly into the power supply to make holistic measures in clusters with multi-GPUs.

References

[1] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. Williams, and K. Yelick. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. Peter Kogge, Editor & Study Lead, 2008.

[2] Run Rules, Green500. Energy Efficient High Performance Computing Power Measurement Methodology. Version: 1.2RC2, 2014.

[3] Jack J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Electrical Engineering and Computer Science Department, University of Tennessee (TN 37996-1301), Computer Science and Mathematics Division, Oak Ridge National Laboratory (TN 37831), University of Manchester, 2014.

[4] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[5] Massimiliano Fatica. Accelerating linpack with CUDA on heterogenous clusters. In David R. Kaeli and Miriam Leeser, editors, *GPGPU*, volume 383 of *ACM International Conference Proceeding Series*, pages 46–51. ACM, 2009.

EnergyLB - saving energy while load balancing on HPC systems *

Edson L. Padoin[1,2], Laércio L. Pilla[3], Márcio Castro[3], Philippe O. A. Navaux[1], Jean-François Mehaut[4]
{elpadoin,navaux}@inf.ufrgs.br, {laercio.pilla,marcio.castro}@ufsc.br, jean-francois.mehaut@imag.fr

[1]Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS) – Porto Alegre, RS – Brazil
[2]Department of Exact Sciences and Engineering
Regional University of Northwest of Rio Grande do Sul (UNIJUI) – Ijui, RS – Brazil
[3]Department of Informatics and Statistics
Federal University of Santa Catarina (UFSC) – Florianópolis, SC – Brazil
[4]Grenoble Informatics Laboratory (LIG)
University of Grenoble – Grenoble – France

1. Introduction and Motivation

The main focus of HPC systems has been performance and today HPC community works toward building Exascale systems (i.e., EFlops), which will provide unprecedented computation, allowing to solve even larger scientific problems. Conceiving Exascale supercomputers by scaling the current technology would demand over a GigaWatt of power [7, 6], which is equivalent to the entire production of a medium size nuclear power plant [17]. With this problem in mind, a global research effort has risen to try to break the Exascale barrier while avoiding such high energy consumption. In 2008, specialists alerted on an official DARPA report [10, 2] that the acceptable power budget to reach Exascale is 20 MW. So, an Exascale system has a limit of $\frac{1\ EFlops}{20\ MW}$, i.e., 50 GFlops/W. Tianhe-2, the current number 1 supercomputer in the Top500 rank, produces 33.8 PFlops consuming 17.8 MW, thus Tianhe-2's energy efficiency is 1.91 GFlops/W. Using Tianhe-2 as a reference, we would yet to increase the energy efficiency 26-fold to match the DARPA recommendation. Thus, to get to Exascale we need to find alternatives to solve the power consumption problem [1, 18].

The first challenge to the development of energy efficient HPC systems and applications lies on increasing their performance while reducing their power consumption. Indeed, saving power has become one of the main concerns of HPC community. To build future Exascale systems we need to take into account power demand and energy consumption constraints and ways to improve energy efficiency [1, 18, 14, 13].

A second challenge arises as HPC systems grow in number of processors and this complicates efficiency once that scientific applications may not be easily equally distributed due to dynamic or irregular characteristics. These characteristics are present in several scientific applications and contribute to a reduced energy efficiency on parallel systems, since the most heavily loaded processor determines the application's performance. Several load balancing strategies have been used to improve the load distribution across processors and to achieve an efficient use of all available resources of a parallel machine.

In this context, state-of-the-art research focuses on either power consumption or load balancing strategies separately. Several works have used DVFS (dynamic voltage and frequency scaling), however this technique may cause performance degradation and an increase in execution time. Other works have used load balancing strategies to reduce the overall execution time, and, consequently, save energy. However, these strategies have used only computational load ignoring power demand or total energy consumption.

In this context, we plan to improve the energy efficiency of parallel systems when running load imbalanced applications. We intend to provide new load balancing strategies that collect system information from each node of the parallel machine and collect information from tasks of the application, in order to manage power demand while the application is running.

* This work was partially supported by CNPq, FAPERGS, FINEP, CAPES (under grants 3471-13-6, 5854-11-3, and 5847-11-7) and by the HPC-GA research project, which has received funding from the European Community's Seventh Framework Programme (FP7-PEOPLE) under grant agreement number 295217. This work was developed in the context of LICIA, an associated international laboratory between UFRGS and University of Grenoble.

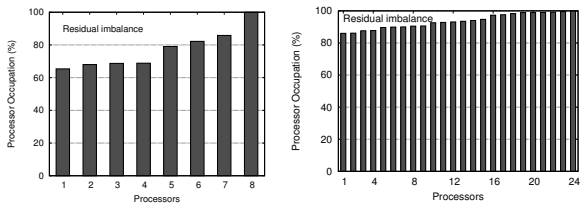
2. ENERGYLB

Reductions in the total execution time are relevant in the perspective of energy consumption, since energy is saved when hardware resources are used for a shorter time. However, it is possible to achieve even greater energy savings if the runtime system is able to exploit the residual imbalances to fine tune the voltage and frequency of cores accordingly. In this case, the challenge lies in reducing the energy consumption of the application while maintaining a similar performance.

In this context, iterative imbalanced applications are potential candidates for energy consumption improvements. These parallel applications are present in different scientific fields. Molecular dynamics, structural dynamics, weather forecast [12], cosmological modeling simulation [4, 8], seismic wave propagation simulations [5], physics [16], and oil exploitation [15], are examples of applications that can benefit from load balancing schemes to improve performance. Our point is that these schemes still leave space for energy optimization.

In an experimental evaluation, tests were performed using 8 and 24 cores (one on each processor) of the parallel machine presented in [13]. We ran a small test with a version of the seismic wave simulator *ondes 3D* [5] using the CHARM++ parallel programming [9, 3].

Load balancers improve the performance of imbalanced applications by making a better load distribution among the available processors. However, they can take sub optimal decisions, which may result in some imbalance remaining. This may happen due to characteristics of the application that prevent a perfectly balanced mapping to be achieved, or due to limitations of the load balancing heuristics, as the problem that they are trying to solve is NP-Hard [11]. In this work we call this remaining load imbalance *residual imbalance*.



(a) 8 processors. (b) 24 processors.

Figure 1. Processor utilization and residual imbalance after the first call of the load balancer GREEDYLB with different processors counts.

Figure 1 shows the residual imbalance when we executed the application with a load balancer named GREEDYLB and different processors counts. In the execution of the application in 8 processors, (Figure 1(a)), we have a residual imbalance of up to 34%. When we ran the application with the same workload in 24 processors, (Figure 1(b)), we obtained a different load distribution, with a remaining residual imbalance of 14% after load balancing. We notice that the residual imbalance is considerably high in several scenarios, as most of the processors are underused. In this case, DVFS techniques can be used to save energy by reducing the processor's frequency and, consequently, their power demand.

In this context, we introduce a new energy-aware load balancing algorithm called **ENERGYLB**, to save energy on iterative applications that present imbalanced loads. Our strategies can reduce the power demand during execution, saving energy and improving the application performance according to its load characteristic. We have implemented our proposed ENERGYLB, using the existing CHARM++ framework, which implements a model of migratable objects [13].

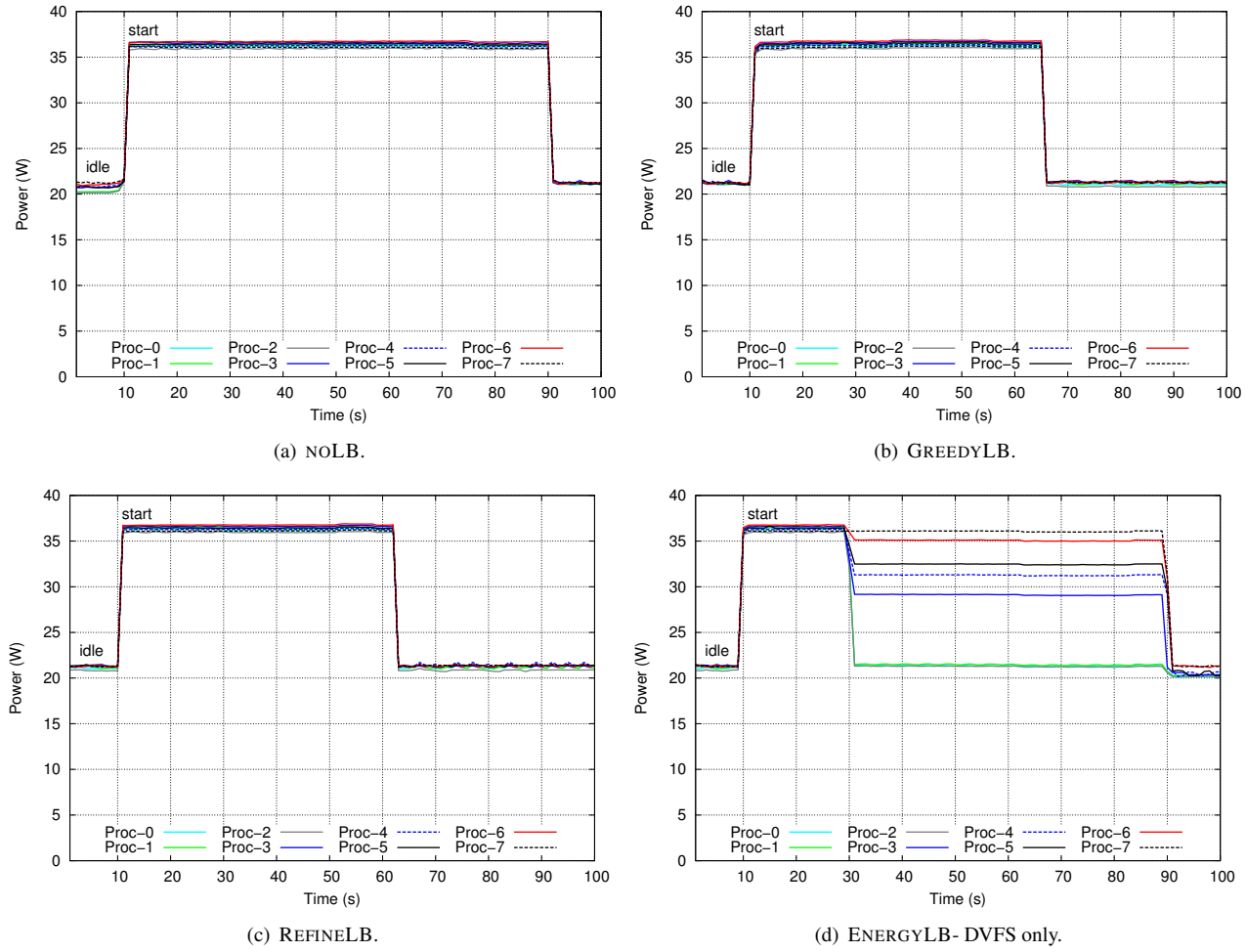
3. ENERGYLB Evaluation

We conducted our experimental evaluation on an SGI Altix UV 2000 system composed of 8 Intel Xeon E5-4640 Sandy Bridge-EP processors with 8 physical cores each. In these processors, there are 14 clock frequency levels available, varying from 1.2 GHz to 2.4 GHz. However, current processors do not allow DVFS on each core individually. So, in this work, tests were performed using 8 cores only, one on each socket.

The operating system used in the machine is a UV2000 GNU/Linux distribution with kernel version 3.0.74. The application benchmark was compiled using gcc version 4.3 and the version 6.5.1 of Charm++. Results show the average of a minimum of 10 runs. The relative error is less than 5% using a 95% statistical confidence with Student's t-distribution.

Our strategy employs a load balancing module that benefits from the load balancing framework available in the CHARM++ runtime system. Aiming to evaluate DVFS only on our ENERGYLB proposal, we disable the load balancing functionality of ENERGYLB load balancer. In this context, we ran *lb_test* benchmark with 500 tasks for 250 iterations.

Figure 2 illustrates the instantaneous power demand measured during the test executions, without a load balancer (NOLB) (representing a baseline execution), using different CHARM++ load balancers (GREEDYLB and REFINELB), and our proposed ENERGYLB load balancer.



(a) NOLB.

(b) GREEDYLB.

(c) REFINELB.

(d) ENERGYLB- DVFS only.

Figure 2. Instantaneous power measured during execution of the imbalanced *lb_test* benchmark using different load balancers.

Figure 2(a) shows the power measured during the execution of the benchmark without any load balancer strategy. When load imbalanced applications are executed, the kernel governor does not realize the imbalance among processors, running the application using the maximum frequency available in the processors, which maintains the power demand constant (always close to 36W in this experiment).

When tests were performed using GREEDYLB and REFINELB, these load balancers perceived the load imbalance and migrated tasks, reducing the total execution time of the benchmark. Still, similarly to the scenario with NOLB, the highest frequency available in each processor was always employed, resulting in a demand of $\approx$ 36W by each processor. Nevertheless, load was not perfectly balanced among all processors, which lead to energy wasting.

When *lb_test* was run with ENERGYLB, all processors executed their task using the maximum frequency during initial iterations, explaining the power demand similar to NOLB. Nevertheless, after calling ENERGYLB, only one processor (the most loaded one) remained at 2.4 GHz. In other words, all others processors had their clock frequencies reduced to intermediate or to the minimum frequency. Consequently, the instantaneous power demand of each processor was also reduced according to their computational load, reducing the power demand. Figure 2(d) illustrates this phenomenon starting at the 20 seconds mark. As the benchmark does not present load variations, no change in the clock frequency of the processors was necessary during other ENERGYLB calls.

The greatest reduction in execution time was achieved when GREEDYLB was used. GREEDYLB load balancer obtain speedup of 1.42 when compared to baseline (NOLB). However, if we compare ENERGYLB results with GREEDYLB, we can observe that the execution

time with ENERGYLB was 1.42 times longer, but the total energy consumption was only 1.19 times greater. This is justified due load imbalance present between tasks. In this test, the largest load difference between processors measured was of 9.2 times and ENERGYLB can only reduce the clock frequency by half (the range the processor goes from 2.4 to 1.2 GHz). Therefore, only adjusting the clock is not sufficient to mitigate the effects of load imbalance in energy consumption.

4. Conclusion

In this paper, we present an evaluation of ENERGYLB employing DVFS only (disabling the load balancing functionality of ENERGYLB). The results achieved show that, when load imbalance was greater than 2 times in the testbed platform, the use of ENERGYLB only is not enough to mitigate processor idleness. Therefore, to avoid this problem, the use of load balancing algorithms together with ENERGYLB is necessary. Due to this effect, we have implemented other approaches that automatically call another load balancer to migrate tasks between processors and then perform DVFS.

We showed that applications implemented in CHARM++ can benefit from ENERGYLB to improve energy efficiency, reducing of power demand during the execution of imbalanced applications, without considerably degrading their overall performance.

References

[1] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. Sancho. Using performance modeling to design large-scale systems. *Computer*, 42(11):42–49, 2009.

[2] P. Beckman, B. Dally, G. Shainer, T. Dunning, S. C. Ahalt, and M. Bernhardt. On the Road to Exascale. *Scientific Computing World*, (116):26–28, 2011.

[3] CHARM++. Parallel programming laboratory (ppl), 2014. http://charm.cs.illinois.edu/.

[4] M. D. Dikaiakos and J. Stadel. A performance study of cosmological simulations on message-passing and shared-memory multiprocessors. In *International Conference on Supercomputing (ISC)*, pages 94–101. ACM, 1996.

[5] F. Dupros, H. Aochi, A. Ducellier, D. Komatitsch, and J. Roman. Exploiting intensive multithreading for the efficient simulation of 3D seismic wave propagation. In *International Conference on Computational Science and Engineering (CSE)*, pages 253–260, 2008.

[6] W. Feng and K. Cameron. The green500 list: Encouraging sustainable supercomputing. *Computer*, 40(12):50–55, 2007.

[7] C.-H. Hsu, W. chun Feng, and J. S. Archuleta. Towards efficient supercomputing: A quest for the right metric. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8–pp. IEEE, 2005.

[8] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, and T. Quinn. Massively parallel cosmological simulations with ChaNGa. In *International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2008.

[9] L. V. Kalé and S. Krishnan. CHARM++: A portable concurrent object oriented system based on C++. In *Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 91–108. ACM, 1993.

[10] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, and e. a. Denneau. Exascale Computing Study: Technology Challenges in achieving Exascale Systems. *DARPA IPTO*, pages 1–297, 2008.

[11] J. Y. T. Leung. *Handbook of scheduling: algorithms, models, and performance analysis*. Chapman & Hall/CRC, 2004.

[12] J. Michalakes and M. Vachharajani. Gpu Acceleration of Numerical Weather Prediction. *Parallel Processing Letters*, 18(04):1–8, 2008.

[13] E. L. Padoin, M. Castro, L. L. Pilla, P. O. A. Navaux, and J.-F. Mehaut. Saving energy by exploiting residual imbalances on iterative applications. In *21st International Conference on High Performance Computing (HiPC)*, pages 1–10, Goa, India, 2014.

[14] E. L. Padoin, L. L. Pilla, F. Z. B. Boito, R. V. Kassick, P. Velho, and P. O. A. Navaux. Evaluating application performance and energy consumption on hybrid CPU+GPU architecture. *Cluster Computing*, 16(3):511–525, 2013. 10.1007/s10586-012-0219-6.

[15] J. Panetta, T. Teixeira, P. R. de Souza Filho, C. A. da Cunha Finho, D. Sotelo, F. M. R. da Motta, S. S. Pinheiro, I. P. Junior, A. L. R. Rosa, L. R. Monnerat, L. T. Carneiro, and C. H. de Albrecht. Accelerating Kirchhoff Migration by CPU and GPU Cooperation. *21st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2009)*, pages 26–32, 2009.

[16] J. Shiers. The worldwide lhc computing grid (worldwide lcg). *Computer physics communications*, 177(1-2):219–223, 2007.

[17] M. Wehner, L. Oliker, and J. Shalf. A Real Cloud Computer. *IEEE Spectrum*, 46(10):24–29, 2009.

[18] A. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers. Efficient resource management for cloud computing environments. In *International Green Computing Conference (IGCC)*, pages 357–364. IEEE Computer Society, 2010.

Proposta de um Formato Binário para Rastro Pajé

Vinícius A.Herbstrith, Lucas Mello Schnorr
{vaherbstrith, schnorr}@inf.ufrgs.br
Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

Resumo

Este artigo apresenta um formato binário para arquivos de rastro em Pajé, almejando principalmente a otimização da leitura através da estrutura binária, de formar a proporcionar um ganho de desempenho para ferramentas que fazem uso do formato para análise e visualização de rastros. A proposta inclui a implementação de dois conversores entre o formato textual e binário e um componente que promove a leitura do arquivo de rastro binário integrado na ferramenta de visualização `PajeNG` e os ganhos em desempenho da leitura do formato binário em relação ao formato textual.

1. Introdução

Com a difusão de aplicações paralelas e distribuídas surge a necessidade da criação de ferramentas que auxiliam na depuração e análises de desempenho destas aplicações. Para tal, se faz a seleção de eventos representativos que devem ser registrados durante a execução em arquivos de rastro de execução [7]. Dentre as informações que compõem estes eventos, temos identificação, tempo do registro, estados de variáveis e outros dados que sejam de interesse para a análise da aplicação em questão.

O formato de rastro Pajé é largamente utilizado para a análise e visualização de rastros de aplicações. O atributo de destaque do formato é a extensibilidade, possibilitando seu uso nos mais variados ambientes de execução, com demandas e requisitos diferenciados, sem ser necessário modificar componentes internos da ferramenta Pajé. Como ponto negativo do formato, temos a sua representação textual, que implica em impactos negativos no desempenho tanto no momento de registro de eventos, gerando uma maior intrusão, como durante a leitura dos arquivos de rastro enquanto processados para a sua visualização e análise. Com o crescente aumento do poder de paralelismo dos computadores de hoje em dia [2], combinado com aplicações mais complexas que geram um número muito grande de eventos durante sua execução, os pontos negativos do Pajé acabam se sobressaindo ainda mais, apresentando uma fraca escalabilidade.

Este artigo apresenta um formato binário para o arquivo de rastro Pajé, como forma de obter uma otimização na leitura de arquivos Pajé. A seção 2 trata da geração dos arquivos em binário, fazendo uso da biblioteca de rastros libRastro [1] e Poti [6], e a implementação da leitura destes arquivos na ferramenta de visualização de rastros PajeNG [5]. A seção 2.1 apresentam com mais detalhes a representação dos eventos Pajé em seu novo formato binário. Por fim, a seção 3 apresenta os resultados dos experimentos feitos sobre a leitura de arquivos de rastros e a comparação dos tempos do formato binário contra a representação Pajé textual padrão.

2. Proposta e implementação

A proposta inclui a implementação de dois conversores e um componente de leitura binário para a ferramenta de visualização PajeNG. Estes três componentes são detalhados na Figura 1 em pontilhado, onde temos os dois conversores (*paje2rst* e *rst2paje*), desenvolvidos dentro da biblioteca Poti com a linguagem C, e o leitor binário (*PajeRastroReader*), desenvolvido na ferramenta PajeNG em C++. Os outros componentes são aqueles que já fazem parte da PajeNG.

As informações do arquivo binário são salvas usando a biblioteca LibRastro, uma biblioteca de geração de arquivos de rastro. A LibRastro tem como principais características uma baixa intrusão na coleta do rastro e flexibilidade nas definições de eventos, características estas que coincidem com o nosso objetivo e requisitos do formato Pajé. Temos então, um arquivo de rastro gerado na LibRastro que contém informações equivalentes ao rastro original no formato Pajé.

A biblioteca Poti – responsável por implementar o formato de rastro Pajé – foi utilizada para a implementação da criação do rastro no formato binário, usando como suporte a libRastro. Foram feitas modificações na biblioteca com o objetivo de gerar a mesma saída Pajé da Poti no

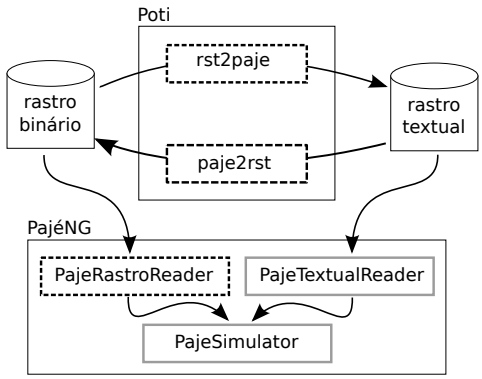


Figura 1. Componentes Poti implementados.

novo formato binário proposto. É possível gerar o arquivo com dois métodos distintos. Um deles é a tradução direta do arquivo textual Pajé fazendo uso da libRastro, nomeado `standard`. O segundo método, chamado `reference`, busca obter uma otimização quanto ao tamanho do arquivo gerado através do uso de referências do tipo int short para dados do tipo string que costumam se repetir durante o arquivo de rastro, gerando um arquivo binário de tamanho reduzido.

Como a execução de aplicações em sistemas paralelos e distribuídos é uma tarefa custosa, é de grande interesse promover ao usuário a possibilidade de converter rastros gerados anteriormente. Para isto, foram criadas duas ferramentas dentro da Poti: `rst2paje` e `paje2rst`.

A ferramenta `paje2rst` faz a tradução de arquivos do formato textual Pajé gerados com a Poti para o novo formato binário. Na implementação deste, foram usados Flex [3] e Bison [4], para fazer a análise léxica e sintática da entrada textual, e as funções da nova biblioteca Poti para gerar o arquivo de saída em formato binário equivalente.

A ferramenta `rst2paje` faz a tradução no sentido oposto, dado o arquivo binário gerado na Poti, é gerado o arquivo no formato textual equivalente. Para este foram usadas as funções de leitura de arquivos da própria libRastro juntamente com funções que fazem a organização dos dados de eventos de acordo com o formato textual.

A leitura e análise do novo formato é feita dentro da PajeNG, uma ferramenta de visualização de rastros no formato Pajé. O componente PajeRastroReader foi criado para realizar a decodificação dos arquivos no formato binário proposto, tanto na versão `standard` quanto na versão `reference`.

Para se tirar proveito da estrutura binária do formato, também foi criado o componente PajeRastroTraceEvent que armazena os dados de acordo com os tipos de dados usados na libRastro(int, doubles e char). O objetivo deste componente é eliminar conversões de dados para o formato string, tarefa essa que se torna muito custosa tendo em vista o grande número de eventos que ocorre em um arquivo de rastro. Outra tarefa da classe PajeRastroTraceEvent é a reordenação dos dados dos eventos, que se torna necessária uma vez que a libRastro faz o armazenamento dos dados em vetores de acordo com o tipo de dado, perdendo a relação de ordem que existe entre o evento Pajé e sua definição de cabeçalho, o que nos leva a ter uma decodificação diferenciada.

2.1. Representação em binário do arquivo Pajé

Um arquivo de rastro Pajé se divide em duas partes: definições de cabeçalho e eventos. As definições de cabeçalho contêm uma lista de tipos de eventos a serem registrados e seus parâmetros, definindo números de identificação, nome e os dados que são usados como parâmetros. Logo após o cabeçalho, temos os eventos datados que foram registrados durante o rastro do programa, com o número de identificação do tipo de evento e os parâmetros, de acordo as regras definidas no cabeçalho.

A libRastro, por ser uma biblioteca de criação de rastros, faz todos os seus registros como eventos, onde cada evento possui um valor numérico para identificação do seu tipo, seguido de vetores com os dados que foram registrados.

Abaixo temos um exemplo de uma definição de cabeçalho de um arquivo de rastro Pajé em seu formato textual.

```
%EventDef PajeDefineContainerType 0
%    Alias string
%    Type string
%    Name string
```

Abaixo a versão binária do mesmo trecho de cabeçalho.

```
type: 999
 strings-> {0}{11}{0}{3}{0}{2}{0}
```

No campo `type` é usado o valor 999 para identificar, durante a leitura, que o evento binário deve ser tratado como uma definição de cabeçalho. As informações de nome de campo e tipo de dado do cabeçalho Pajé são representadas por valores inteiros na versão binária, valores estes definidos em uma enumeração que faz parte da `PajeNG` e da Poti. No exemplo, o primeiro valor do vetor de dados é a identificação do evento, que é o mesmo valor usado na representação textual (0, no exemplo), seguido por tuplas nome do campo e tipo de dado. No exemplo, a primeira tupla da sequência é Alias e string, representadas no arquivo binário pelos valores 11, o valor da enumeração para o campo Alias, e 0, valor da enumeração para o tipo de dado `string`. Os quatro valores seguintes do vetor de dados em

binário correspondem às outras duas tuplas, com os valores 3 e 2 representando os campos Type e Name respectivamente e 0 o tipo string.

Abaixo, um exemplo de um evento datado no formato Pajé textual, evento este que corresponde à definição de cabeçalho anterior.

```
0 1 0 HOST
```

Abaixo temos a representação com a libRastro para um evento datado obtido com o conversor `paje2rst` com o método `standard`.

```
type: 0
 strings-> {1} {0} {HOST}
```

Diferentemente da definição de cabeçalho, em eventos a informação do campo `type` se refere à identificação do evento, com o valor 0 no exemplo indicando que o evento é do tipo `PajeDefineContainerType`. Em seguida, temos os vetores com os dados do evento datado agrupados de acordo com o seu tipo, neste caso todos em um vetor de strings.

A seguir temos a representação do mesmo evento datado obtido também através do `paje2rst`, mas usando o método `reference`.

```
type: 888
 strings-> {HOST}
type: 888
 strings-> {0}
type: 888
 strings-> {1}
type: 0
 u_int16_ts-> {2} {1} {0}
```

Os eventos em que o campo `type` tem o valor 888 definem que o evento contém uma string à ser usada como referência. No evento de exemplo temos três dados do tipo string, logo temos três eventos de referência que armazenam as strings. Abaixo destes eventos existe o evento Pajé de fato, que faz referência às strings que são indicadas de acordo com a ordem em que elas aparecem no arquivo como um valor inteiro de 16 bits.

Embora um número maior de eventos seja gerado, temos uma redução significativa do tamanho total do arquivo com este método devido a grande ocorrência de dados do tipo string.

3. Resultados sobre o desempenho do `PajeRastroReader`

A análise de desempenho da leitura do formato binário foi feita sobre dois arquivos de rastro de duas aplicações distintas, cada um em três versões, textual – a versão original – , binária e binária reduzida – ambas obtidas com a ferramenta paje2rst. Uma das aplicações é uma fatoração LU que gerou um arquivo de rastro de 0.77 GB, a outra é um cálculo de gradiente conjugado com um rastro de tamanho 2.1 GB. A máquina usada para os testes possui processador Intel i7-4770 CPU (3.40GHz), HD 1TB SATA II (3.0Gb/s) e 8GiB DIMM DDR3(1.6GHz). Foram feitas 30 execuções para cada versão, binárias e textual, dos dois arquivos, de forma a se obter uma amostra significativa para comparação sobre o desempenho do formato proposto.

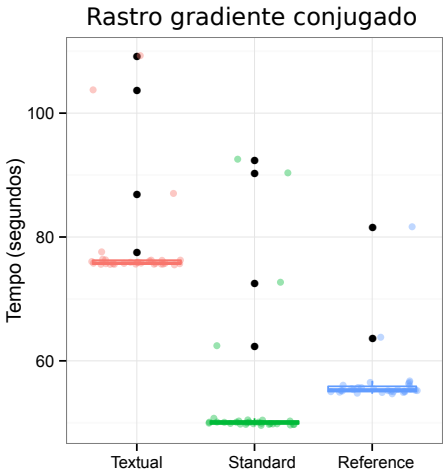


Figura 2. Resultados dos experimentos com o arquivo de rastro gradiente conjugado

Em ambos os gráficos, podemos notar um ganho de desempenho de aproximadamente 32% do formato binário `standard` em relação ao formato textual original. Com a versão `reference` o ganho foi de 27% em relação ao formato textual. Sobre o tamanho do arquivo, a diferença de tamanho entre o arquivo original e o do método `reference` foi de 48% para o arquivo de rastro da fatoração LU e 43% para o rastro do gradiente conjugado.

4. Conclusões

Neste artigo apresentamos o formato binário Pajé. Como esperado, a estrutura do arquivo binário promoveu uma melhora no desempenho da leitura do arquivo de rastro. Esperava-se ganhos maiores com a redução do tamanho do arquivo usando o método `reference`, mas o tempo de processamento das referências acaba sendo maior que a simples leitura dos dados em binário com o método `standard`. Ainda sim, ambos os métodos de geração de arquivos resultam em um ganho de desempenho de leitura sobre o formato textual, reduzindo em até 32% o tempo de leitura para arquivos gerados com o método `standard`.

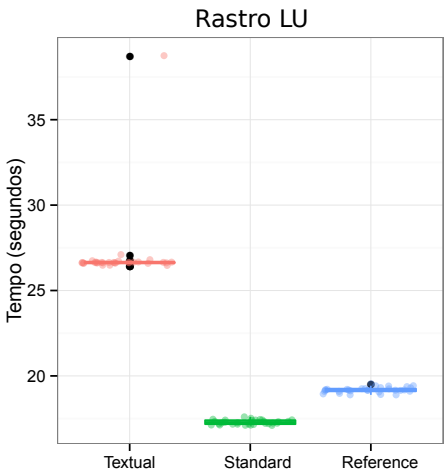


Figura 3. Resultados dos experimentos com o arquivo de rastro LU

A partir destes resultados positivos, podemos buscar ainda outras otimizações para o formato binário. Como possibilidade, temos o uso de técnicas de compactação que tornariam o arquivo de rastro menor e consequentemente um melhor tempo de leitura com técnicas de descompactação eficientes aplicadas durante a leitura do arquivo binário.

Referências

[1] G. J. da Silva, L. M. Schnorr, and B. Stein. Jrastro: A trace agent for debugging multithreaded and distributed java programs. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, pages 46–54. Los Alamitos: IEEE Computer Society, 2003.

[2] T. Geller. Supercomputing exaflop target. In *Comm. of the ACM*, page 54(8). ACM New York, NY, USA, August 2011.

[3] V. Paxson. Flex. `http://flex.sourceforge.net`, 2014.

[4] T. G. P. Robert Corbett. Bison. `http://www.gnu.org/software/bison`, 2014.

[5] L. M. Schnorr. Pajeng. `http://github.com/schnorr/pajeng`, 2014.

[6] L. M. Schnorr. Poti. `http://github.com/schnorr/poti`, 2014.

[7] B. Stein. Depuração de programas paralelos. In *I Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul*, pages 151 – 176, Gramado, RS, 2001. Sociedade Brasileira de Computação (SBC).

Trace comparison using a sequence alignment algorithm

Alef Farah, Lucas Mello Schnorr
PPGC, Institute of Informatics, UFRGS
Bento Gonçalves Avenue, 9500, Porto Alegre, RS, Brazil

Abstract

The comparison of execution traces enables an in depth analysis of the differences between two versions or runs of a parallel application. The automatic comparison of possibly huge trace files poses challenges on automation, scalability, and data visualization. In this article we present a strategy to overcome these difficulties based on a sequence alignment algorithm used in bioinformatics. We find the development of a scalable tool based on this strategy to be plausible. However, the initial experiments we executed led us to question if the technique in itself provides meaningful information for the analyst.

1. Introduction

Application tracing and trace visualization enable the developer of parallel and distributed applications to examine his software in detail, in order to perform the fine-tuning often necessary while developing for complex environments. Visualization techniques coupled with data aggregation methods also allow an overview of the application behavior [2], which is often desired when the amount of registered events is overwhelming.

The developer might also wish to compare different versions of his application to identify performance loss or improvement within the modified parts of his program. It is also desirable to evaluate an application behavior in different architectures or networks, and comparing its execution traces is one way to identify both subtle and noticeable differences. The standard approach still is a manual comparison of the visualization of the trace files, leaving the pattern recognition work for the human brain. With the growth both in size and complexity of data and code, an automatic comparison is often not only desirable, but imperative.

Among the difficulties of an automatic comparison is the identification of similar regions between two programs with different execution times for each function call, possibly different order for asynchronous tasks and communications, and so forth. Performing the visualization of an execution trace in a scalable way is no trivial task with current data volumes. Besides being scalable performance-wise, the application also needs to render a comprehensible visualization for the user even with huge execution traces.

We tested proposed solutions for the aforementioned difficulties based on sequence alignment and data aggregation techniques. We implemented the former and experimented with the latter to evaluate their usefulness and feasibility, seeking possible improvements.

The rest of this paper is organized as follows. In section 2 we present the strategies we studied. In section 3 we present an overview of their implementation, and in sections 4 and 5 we show initial results on their usefulness and feasibility, respectively. Section 6 presents a summary of our contributions.

2. Related work

P. Velho, L. M. Schnorr, A. Legrand and H. Casanova [10] use markers to identify comparable regions, a strategy present in various tools such as the popular Vampir visualization tool [1]. However, this approach as it is currently used lacks automation as markers need to be added manually by the analyst. M. Weber et al [11] borrow a classic algorithm used in bioinformatics to do a pair-wise alignment of two trace files. They couple it with custom similarity metrics [12] and use a proprietary visualization tool [1] to display the results. We reproduced this strategy using free software tools such as Pajé [7], and evaluate the validity of our implementation in sections 4 and 5. Dosimont, Damien et al [2] use a data aggregation technique to summarize regions of large execution traces. We have experimented with aggregating data from two comparable trace files instead of aggregating comparable events within a single file, and present initial results in section 4.1.

In our work we studied the sequence alignment and data aggregation techniques, re-implementing the former and experimenting with the latter. Our objective was to reevaluate them and explore new possibilities, focusing on innovative visualization techniques for performance analysis of parallel applications.

3. Methodology

Aligning similar regions of two sequences allows the user to compare them more easily. More so, a similarity score can be calculated and used with visualization techniques to highlight these regions. This strategy, which belongs to a wide family of algorithms, is used to compare strings (edit distance, longest common subsequence), DNA sequences (Needleman-Wunsch [8], Smith-Waterman [9]), and in our case, sequence of events registered in two comparable trace files. The alignment of similar regions of the sequences can be thought of as shifting the elements in order to pair up those that are the same in both sequences.

To uniquely identify each event in the sequence we use its identifier from the trace file, the one already provided by the Pajé trace file format. For instance, for function calls the identifier is usually the function's name. Since execution times are different even between two executions of the same application in the same machine, event timestamp is not taken into consideration for comparison, only the order in which the events have occurred. This is the same idea used in the work in study [11].

To align the sequences we have used a modified version of Hirschberg's algorithm for finding the longest common subsequence between two strings [6], much like it is used in bioinformatics as a space linear version of the Needleman-Wunsch algorithm, used to align DNA sequences. The algorithm is quadratic in time; the procedure is as follows.

3.1. Implementation

While comparing the events in the sequence, we attribute the scores α for events that are equal, β for events that differ, and γ if an event is present only in one sequence, in which case we say a gap has occurred. The numerical value for each score can be defined by the user, the default being $\alpha = 2, \beta = -1, \gamma = -1$, penalizing differences and rewarding similarities. A scores matrix D is recursively filled for every element in the sequences A and B, which represent the sequence of events from matching execution flows from two trace files. This step is done for every execution flow present in both execution traces. One extra row and column are added at the beginning and initialized as follows. Let $\theta(i, j)$ be a function that returns α or β comparing two events:

$$D(i,j) = \begin{cases} j * \gamma & i = 0 \\ i * \gamma & j = 0 \\ max \begin{cases} D(i-1, j-1) + \theta(i,j) & \\ D(i, j-1) + \gamma & i \neq 0, \\ D(i-1, j) + \gamma & j \neq 0 \end{cases} \end{cases}$$

For each cell the score is calculated using the cell above it, to the left, and both (diagonal). Because of that, only two rows need to be kept in memory at any given time. This enables a space linear implementation of this algorithm.

This step is done recursively, dividing the problem into two smaller ones until a trivial case is reached, to which we apply the classical Needleman-Wunsch algorithm for aligning the sequences. The alignment is done by tracebacking in the matrix and finding a "shortest path" (i.e. passing through the cells with maximum similarity score) from the last cell to the first.

3.2. Parallelizing the algorithm

We first tried to parallelize the algorithm previously described to overcome its quadratic time. We use the call stack of the execution traces to determine which sub-sequences of events – i.e. which execution flows – can be compared in parallel. Since events from one execution flow are never comparable to events from different execution flows they can be aligned in parallel, as suggested in the work being studied [11].

Besides parallelization, a suggested optimization is to stop traversing a sub-tree when there is a difference in two events. In the green sub-sequences from Figure 1, events $3, 4$ and C, D differ and their children would not be compared, but marked as different. This can be switched on or off by the user according to his needs. Notice that in Figure 1 there are events missing from the purple execution flow when compared with the matching execution flow from the other trace. In this case, events representing a gap would be added in order to fill the void.

4. Alignment visualization

We opted to use existing visualization tools and do so by writing the aligned sequences to new trace files. This allows the analyst to use whatever tool he desires, and also has the advantage that an alignment is only performed once. If the user wishes to revisit a previous alignment, he simply loads the already aligned trace files to his favorite visualization tool.

Figure 2 shows a Gantt chart of two unaligned execution traces. On top, the execution trace of an application parallelized with OpenMP using a "dynamic" scheduler, consisting of a loop calling a "do work" function. Below, the same application using a "guided" scheduler. Green polygons represent the "do work" events and red polygons the implicit barrier at the end of the loop. Events are displayed along the horizontal axis, which represents time. Each row represents a different execution flow. The Pajé trace visualization tool [7] was used to generate these representations. Manually comparing the visualizations it is clear that the

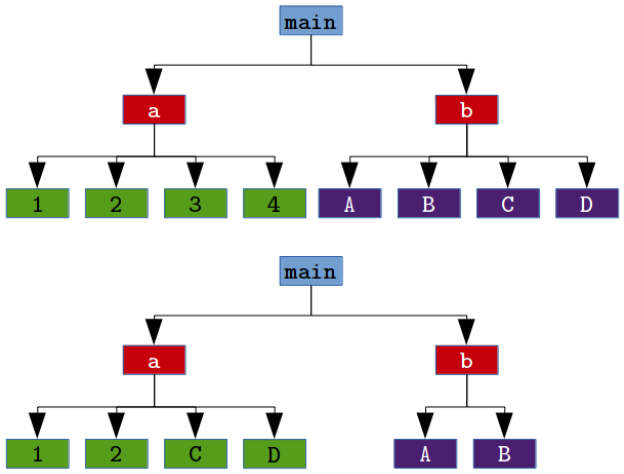


Figure 1: Visual representation of calling context trees for two trace files, where each polygon represents an event. The order in which the polygons appear in the figure (left to right) represent the order they occur in the trace file. Matching execution flows have the same color.

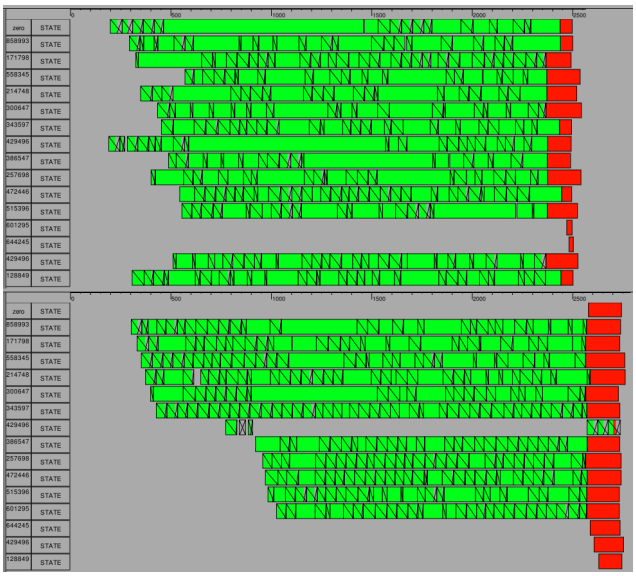


Figure 2: Unaligned execution traces.

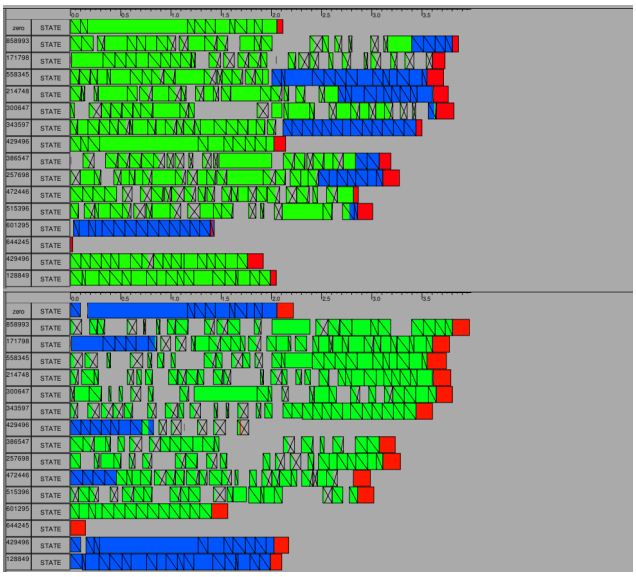


Figure 3: Aligned execution traces.

dynamic scheduler finished first and had a better distribution of the tasks among the threads.

Since during the alignment we disregard the timestamp from each event and insert gaps when one event is present only in one sequence, we end up loosing the original time information. In other words we shift events in time in order to align similar ones.

We visualized the alignment of simple execution traces and reproduce some results below. For the cases we studied, we found that the time shifting resulted in too much information loss, and we put the value of the shifted execution traces in question. For our test cases, we did not found that the alignment of similar regions provided enough benefits to justify the loss of timing information and meaning.

Figure 3 shows the aligned execution traces. Additional blue polygons represent gaps inserted due to an event being present solely in the other execution trace. Chronological information is lost so we can no longer easily determine which one had the shortest execution time. More so, we cannot easily determine how the tasks were distributed among the threads.

To overcome the issue with the loss of timing information, the authors of the original study propose several metrics and visualization techniques to indicate the shifts in time. Without them, we question the usefulness of the alignment strategy for certain cases. For the ones we studied, the manual comparison of the original trace visualizations proved to be better than the automatic approach.

4.1. Data aggregation

We also experimented with data aggregation techniques, more specifically with the Ocelotl Analysis Tool, which uses a custom spatiotemporal aggregation method [2]. We evaluated the possibility of comparing execution traces by aggregating regions that are homogeneous between them. At first, we merely merged two trace files into one and let the tool aggregate regions as it saw fit. Again, preliminary results shows that the information loss (this time due to the

aggregation of incomparable regions) was too big, and the visual result of the merge was unsatisfactory.

An idea to overcome this issue is to merge the sequence alignment and data aggregation techniques. Because of the aforementioned difficulties with the stand-alone techniques, we postponed these tests until we resolve the issues we're having – possibly with similarity metrics and alternative visualization techniques.

5. Performance and scalability

We evaluated the performance of our application with artificial trace files of predetermined size generated by a script, and later with a few test cases [4]. Although more tests are necessary, our results were close to the ones obtained by previous works [11], both in total execution times and in the trend with which they grow as the file size increases – less than quadratic. Execution times for small files are seconds or less, and 2.5GB files with millions of events are aligned in less than two minutes in a personal computer [5].

In order to ascertain anything about the scalability of the tool, we have yet to test it with large scale traces and perform an in depth analysis. So far we obtained satisfactory results for small and medium trace files.

6. Conclusion

The preliminary tests we ran lead us to find that for the sequence alignment strategy a metric to indicate the time shifting is imperative. For our test cases, the alignment in itself does not seem to provide the analyst with enough information, and can cause confusion about the meaning of events shifted in time.

As to the performance and scalability of the alignment algorithm, our benchmarks show that it has reasonable execution times with medium trace files. The fact that the execution times rise in a less than quadratic way makes it plausible to think about a scalable tool based on this strategy.

We have some suggestions as to the alignment algorithm, such as using different alignment types, namely a semi-global alignment [3]. We are also working in a hybrid parallelization approach. However, until the visualization issues have not been solved, those suggestions will have to wait. Among the next steps in our research are additional test cases for the alignment visualization, specially with more complex applications, and the combination of the alignment and aggregation techniques. We are concurrently moving in a different direction and analyzing the intrusion of different tracing applications.

7. Acknowledgments

The authors would like to thank the National Counsel of Technological and Scientific Development (CNPq) for the financial support for the project.

References

[1] H. Brunst, D. Hackenberg, G. Juckeland, and H. Rohling. Comprehensive performance tracking with vampir 7. In M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 17–29. Springer Berlin Heidelberg, 2010.

[2] D. Dosimont, R. Lamarche-Perrin, L. Schnorr, G. Huard, and J.-M. Vincent. A spatiotemporal data aggregation technique for performance analysis of large-scale execution traces. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*, pages 149–157, Sept 2014.

[3] S. R. Eddy. A probabilistic model of local sequence alignment that simplifies statistical significance estimation. *PLoS computational biology*, 4(5):e1000069, 2008.

[4] A. Farah and L. M. Schnorr. Comparação de rastros para análise de desempenho de aplicações paralelas. *Anais da XV Escola Regional de Alto Desempenho, pp. 201-204*, 2015.

[5] A. Farah and L. M. Schnorr. Comparação de rastros para análise de desempenho de aplicações paralelas (slides). http://erad2015.inf.ufrgs.br/downloads/p/ic/s8-1.pdf, 2015.

[6] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM 18 (6): 341–343*, 1975.

[7] Kergommeaux, Stein, and Bernard. Pajé visualization tool. Parallel Computing, 26(10):1253–1274, 2000.

[8] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[9] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[10] P. Velho, L. M. Schnorr, H. Casanova, and A. Legrand. On the validity of flow-level tcp network models for grid and cloud simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 2013.

[11] M. Weber, R. Brendel, and H. Brunst. Trace file comparison with a hierarchical sequence alignment algorithm. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 247–254, July 2012.

[12] M. Weber, K. Mohror, M. Schulz, B. Supinski, H. Brunst, and W. Nagel. Alignment-based metrics for trace comparison. In F. Wolf, B. Mohr, and D. Mey, editors, *Euro-Par 2013 Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 29–40. Springer Berlin Heidelberg, 2013.

PajeNG Multi-trace File Parallelization Strategy

Jonas H. M. Korndorfer
Email: jhmkorndorfer@inf.ufrgs.br

Lucas Mello Schnorr
Email: schnorr@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul
Instituto de Informática

Abstract

Trace-based observation of a parallel program execution is one of the most helpful techniques for performance analysis. This approach frequently has to deal with large traces, which creates a great challenge: how to process and analyze such information in an efficient way. We address this challenge using PajeNG, a performance analysis toolset. In its current form, PajeNG is sequential and fails to scale for large trace files. Paper presents a parallel strategy for PajeNG, together with a preliminary performance analysis. The experiments executed on a NUMA machine show that we scale linearly up to four threads with good efficiency.

1. Introduction

The current increasing demands of computing power are tending for the development of parallel applications that better use the multi-core architectures existing today [8]. Such development is a complex task, affected by at least two factors: the first is the high scalability of current supercomputers; the second is the lack of deterministic execution since processing nodes are independent. Two executions with the same input can demonstrate completely different behavior. Some libraries like OpenMP [1] and MPI [4] are used to help the development of these parallel applications. However even with supporting libraries, the success of the development of a high performance parallel application depends on a correct mapping of the available resources.

The behavior understanding of the application execution is one of the most powerful methods to identify potential bottlenecks and misused resources. Frequently this is done by tracing an application collecting performance counters and important events during the execution. This method generally results in large trace files reaching the order of gigabytes of raw performance data. The processing of this data must be efficient to conduct a fruitful performance analysis.

Over the years, several tools were developed to help the performance analysis of parallel applications. Most of these, like Vampir [6] and Scalasca [2], are focused on the processing of traces with specific semantics which targets libraries like OpenMP and MPI. A minor part of these, like PajeNG [10], have its own trace file format without associated semantics. This work focuses on the optimization of PajeNG on its task of processing trace files. The great advantage of this tool is that with its trace file format without semantics [9], it can be used to represent data coming from several different applications without any modification. As of today, PajeNG trace processing is sequential, so large amounts of data spend much time to be processed. The work described in this paper presents a parallelization strategy for PajeNG intending to improve its performance when dealing with very large trace files.

This text is structured as follows. Section 2 details the sequential PajeNG. Section 3 explains how our parallelization strategy works and the modifications applied to PajeNG. The Section 4 presents a preliminary performance analysis of our strategy. Section 5 mentions some works/tools similar to ours. Finally we conclude in Section 6 with a discussion and suggestions for future work.

2. Original PajeNG Implementation

PajeNG [10] is a trace-based tool-set for performance analysis developed in C++. It works with its own trace file format and provides four tools/libraries with different proposals: *pj_validate*, *pj_dump*, *pajeng* and *libpaje*. *pj_validate* is used to analyse the integrity of the trace data, outputting some statistics. *pj_dump* can export the results of the processing in a CSV-like file format. *pajeng* is a visualization tool that generates interactive space/time representations of the traces. Finally *libpaje*, which is the main focus of this work, is an open source library that does the trace processing in all other tools contained in PajeNG.

2.1. Description of the Pajé Trace Format

This subsection describes the Pajé trace file format. We have to include these concepts here because they are very important for the comprehension of how PajeNG and our strategy work. The Pajé trace file format is self-defined, textual, and generic. The genericity allows anyone to describe the behavior of the execution of any kind of parallel and/or distributed systems whose behavior can be described by timestamped events. The traces in the Pajé format can be one of five types:

Container can be any monitored entity like a network link, a process, or a thread. It is the most generic object and the only one that can hold others, including other containers;

State represents anything that has a beginning and ending time-stamp. For example: a given container starts blocked and then, after some time, changes its state to free. This period of time is a state;

Variable represents the evolution of the value of a variable along time. A common example is the number of exchanged messages along time;

Link can be used to represent the causality relation between two containers having a beginning and ending timestamps. The traditional example is a message passing between two processes;

Event is anything remarkable enough to be uniquely identified and has only one time-stamp.

2.2. Workflow Features & Problems

libpaje is responsible for the trace replay, working with one trace file each time through a sequential processing. This is done by three main classes: *PajeFileReader*, *PajeEventDecoder* and *PajeSimulator*. Figure 1 illustrates *libpaje*'s workflow, from left to right. *PajeFileReader* reads chunks of the trace file, sending them to the second class. In the next step, an object of the *PajeEventDecoder* class receives the chunks and splits them into lines which are transformed in generic event objects. The flow of events generated by the decoder are received by the *PajeSimulator*. For each event, this component simulates the event behavior changing the current data of each container. The history of changes generated by all events is kept in memory by the simulator.

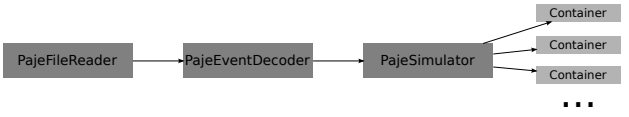


Figure 1. Original PajeNG's workflow.

This processing model has performance problems. When the trace files become larger the replaying time is extended in a prohibitive way. Figure 2 shows a simple example: in the X axis are placed different trace file sizes and in the Y axis the average run time in seconds thirty executions. In this we can observe that the time spent increases linearly according to the size of the files. So, given that massively parallel applications often produce huge traces, this becomes a serious issue for the tool.

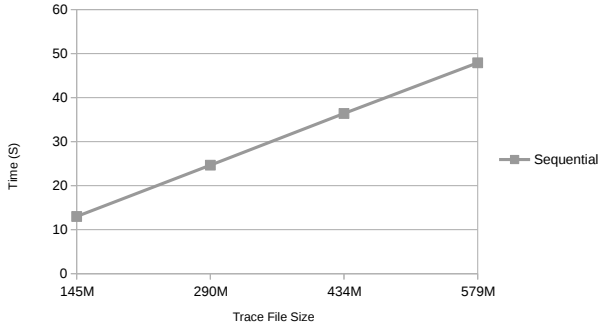


Figure 2. Sequential spent time to process different trace file sizes.

3. PajeNG Parallelization

The strategy used for the parallelization of the PajeNG intends to maximize the amount of data read from the disk. This approach was chosen due to our previous work [5] which has noted that most of the time spent by the tool is led by the first two stages of the workflow: *PajeFileReader* and *PajeEventDecoder* section 2.2. The solution described here comprises two changes: a change in the Paje file format to allow multi-file traces subsection 3.1; and the adoption of C++11 threads in the libpaje workflow subsection 3.2.

3.1. Pajé Trace file Modifications

The support for multiple trace files was made by modifications in the Pajé trace file format. Such modifications consist in creating a new event type called *PajeTraceFile* and splitting the trace file in several pieces. The informations contained in this event are: its type, container and a file name or path. These informations will be used by the *PajeSimulator* to start new threads. Therefore the *PajeTraceFile* event works like a mark which indicates where are the remaining informations of a trace file and when to start a new execution flow/thread.

The splitting of the trace file can be done in several ways. For the first experiments of our strategy we adopt the following metric. The trace file was split for each *container* event found. Therefore, each piece contains all informations of one container. Besides was created a starter trace file. This file only stores *container* and *PajeTraceFile* events. So when the *PajeSimulator* performs such trace, it throws several flows in separated threads that will process each one a different *container*.

3.2. Parallel libpaje Workflow

libpaje had to be modified in two ways: first to support the new event type and second to use C++11 threads for the parallel processing. The processing of *PajeTraceFile* events is made by a new class named *PajeParallelProcessor*. This class have implements a queue of tasks and a pool of workers. These workers consist in C++11 threads and the pool size is user defined. The queue of tasks is filled by *PajeTraceFile* events.

The modifications described above end up building a new workflow for the libpaje. The behavior of this workflow is shown in Figure 3. The main thread is presented in the top of Figure 3. The initial steps of its processing are equal to the sequential: starting by the *PajeFileReader* (pFR) sending to the *PajeEventDecoder* (pED) and finally to the *PajeSimulator* (pS), like described in the subsection 2.2. The differences begin when a *PajeTraceFile* event (PTF) arrives in the *PajeSimulator*. In this case the *PajeSimulator* inserts the *PajeTraceFile* event into the tasks queue of the *PajeParallelProcessor* described above (the *PajeParallelProcessor* is not presented in the Figure 3, it is abstracted by the dotted lines). In our strategy, the main thread processes the starter trace file (remember that it is for our trace file splitting approach subsection 3.1). Therefore it works like a producer just filling the queue of tasks.

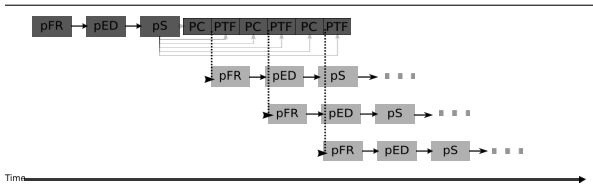


Figure 3. Representation of our parallelization strategy for the PajeNG.

The subsequent steps, right below of Figure 3, have presented the workers which will consume the tasks queue. Each worker withdraws a *PajeTraceFile* event from the queue and according to its information, *container* (PC) and trace file name/path, it launches a new workflow in a separated thread. These new workflows are also equal to the sequential.

4. Experiments and Results

This section presents the results obtained by our experiments with the parallelization strategy for PajeNG described above. We have defined two different configurations for the threads locality: free threads and pinned threads. For both, we have executed thirty executions for each different number of threads: 1, 2, 4, 8 and 16 threads respectively for a total of 270 executions.

The experimental platform is a NUMA-machine with four NUMA nodes, each with eight physical processors Intel(R) Xeon(R) CPU X7550 running at 2.00GHz. There is a total of 128 GB of RAM. The storage system is based on a RAID Dell PERC H700 SATA 3 with 6Gb/s of throughput and with several hard drives attached totalizing 5TB.

Figure 4 shows the speedup obtained by our strategy comparing both versions: pinned version in blue, free version in red and linear speedup in grey. The results in this plot are based on the mean of the execution times. As we can see the speedup is nearly linear until four threads for both versions. For eight threads we note that the speedup decreases significantly in both versions, however the pinned version presents better results. After eight threads the strategy become to loose performance. It probably occurs because of the great concurrency of the threads reading the disc, however we cannot show evidence of that since we are still investigating why we loose so much performance when we reach 8 threads.

5. Related Work

Our work is closely related to prior research in the area of parallel application tracing and profiling [7, 3, 6, 11]. Trace-based tools like Vampir and Extrae collect application traces that are very attached to the semantics of the used programming libraries. They have good scalability but are not dynamic like PajeNG. The most similar tool with PajeNG is Paraver [7] that also has its own trace file format. However this tool achieves high performance by reducing the information in the trace file which can cause information loss.

6. Conclusion

PajeNG is a very useful toolset for the performance analysis of parallel applications. The sequential version loses much of its utility when it has to process large traces. Therefore we have developed a parallelization strategy to better use the capacity of this toolset. The first results are very optimistic with linear scaling PajeNG up to four threads. The

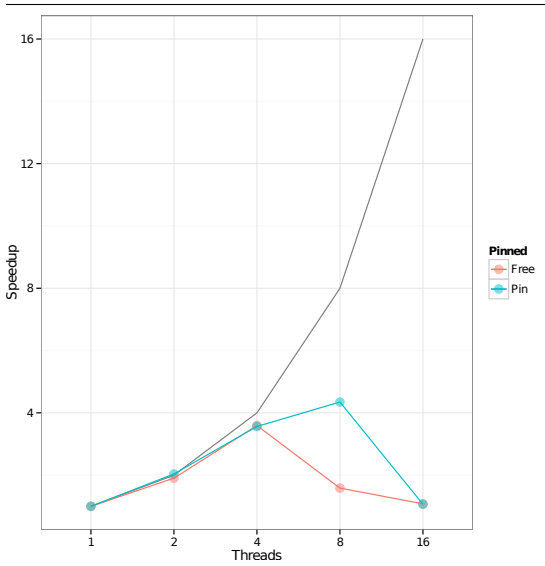


Figure 4. Speedup graph of the multi-trace file strategy for PajeNG.

work is under study and further experiments are being developed to improve even more the scalability. Some of these are: the correct mapping of the threads location and how optimize disk usage. Besides, this implementation creates opportunities for future work, since each thread is independent it can be easily used to the implementation of a distributed version.

References

[1] L. Dagum and R. Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, 1998.

[2] M. Geimer, F. Wolf, B. J. N. Wylie, E. Abraham, D. Becker, and B. Mohr. The scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, 2010.

[3] H. Gelabert and G. Sánchez. Extrae user guide manual for version 2.2. 0. *Barcelona Supercomputing Center (B. Sc.)*, 2011.

[4] W. Gropp, E. L. Lusk, and R. Thakur. *Using MPI-2: Advanced Features of the Message Passing Interface*. The MIT Press, Cambridge, Massachusetts, USA, 1999.

[5] J. H. Korndorfer and L. M. Schnorr. First efforts for the parallelization of pajeng. 2014.

[6] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. Vampir: Visualization and analysis of mpi resources. 1996.

[7] V. Pillet, J. Labarta, T. Cortes, and S. Girona. Paraver: A tool to visualize and analyze parallel code. In *Proceedings of WoTUG-18: Transputer and occam Developments*, volume 44, pages 17–31, 1995.

[8] N. Sadashiv and S. D. Kumar. Cluster, grid and cloud computing: A detailed comparison. In *Computer Science & Education (ICCSE), 2011 6th International Conference on*, pages 477–482. IEEE, 2011.

[9] L. M. Schnorr. Pajé trace file format. Technical report, UFRGS, Porto Alegre, Brazil, 2014. `http://paje.sf.net`.

[10] L. M. Schnorr. Pajeng – paje next generation, 2014. `http://github.com/schnorr/pajeng`.

[11] F. Wolf, B. J. Wylie, E. Abrahám, D. Becker, W. Frings, K. Fürlinger, M. Geimer, M.-A. Hermanns, B. Mohr, S. Moore, et al. Usage of the scalasca toolset for scalable performance analysis of large-scale parallel applications. In *Tools for High Performance Computing*, pages 157–167. Springer, 2008.

Analyzing Performance and Efficiency of HPC Applications in the Cloud

Vinicius Facco Rodrigues, Gustavo Rostirolla, Rodrigo da Rosa Righi, Cristiano André da Costa
Applied Computing Graduate Program, Universidade do Vale do Rio dos Sinos (Unisinos)

E-mail: viniciusfacco@live.com, grostirolla1@gmail.com, rrrighi,cac@unisinos.br

Abstract

Elasticity is a key feature of cloud computing in which the resources can be dynamically changed on-the-fly. In the HPC scenario, a user may want to adjust the resources availability to increase performance. A way to measure performance and efficiency of parallel applications with fixed number of resources is the speedup metric. However this metric does not apply to elastic cloud infrastructures. Thus, this paper proposes new metrics to evaluate performance and efficiency of iterative parallel applications on the cloud. Results show a relationship between performance and efficiency.

1. Introduction

A key feature of cloud computing is elasticity, in which users can scale their computational resources up or down at any moment according to demand or desired response time [5, 11]. Considering the HPC (High-performance Computing) landscape and a very long-running parallel application, a user may want to increase the number of instances to try to reduce the completion time of the application. The success of this procedure will depend on both the computational grain and application modeling [4, 7]. In HPC systems, elasticity can be a double-edged sword involving performance and efficiency. Directly related to both, we have resource consumption, which can also help in measuring elasticity quality. Traditionally, the speedup metric is used to measure performance and efficiency of parallel applications in scenarios where the number of processes is the same in the entire application execution time. However, in elastic environments, the number of processes can change at any moment and the speedup metric may not be suitable.

Our analysis shows that related work presents a gap concerning defining decision functions for elasticity viability in the combination of performance and efficiency, besides not addressing a discussion of lower and upper thresholds [2, 3, 4, 6, 9, 10, 11]. Responding to this motivation, we propose a redefinition of the so-called speedup and parallel efficiency metrics for elastic HPC environments, here denoted as ES (Elastic Speedup) and EE (Elastic Efficiency). In previous work, we developed an elasticity model called AutoElastic [8] to deploy an elastic cloud computing environment, which automatically reorganizes resources for loop-based synchronous parallel applications. AutoElastic acts at the PaaS (Platform as a Service) level of a cloud, hiding any details from users about horizontal elasticity in terms of both application writing and threshold management. Thus, this article presents a performance analysis of HPC applications in an AutoElastic cloud using the proposed metrics. We performed an analysis of a set of lower-upper thresholds and different computational workloads across a CPU-intensive HPC application.

2. Related Work

This section presents some metrics to evaluate cloud systems and applications. Roloff et al. [9] define a Cost Efficiency metric, which is obtained by considering the average performance and financial cost per hour when running HPC applications in the cloud. Martin et al. [6] provide two metrics to analyze a system's behavior: $\frac{cost}{performance}$ and $\frac{cost}{throughput}$. In their work, cost refers to the financial cost of executing an application in the cloud, performance means the makespan to compute a series of tasks, and throughput is the number of tasks completed in accordance with time. Similarly, Coutinho et al. [2] explore $\frac{Cost}{Performance}$ and $\frac{Cost}{Bandwidth}$ rates. In addition to these metrics, they also propose a metric, $\frac{hours}{instances}$, that contemplates the total execution time and the number of instances required. Weber et al. [11] present two metrics to analyze cloud elasticity. The idea consists of comparing the number of scale events (D_u or D_d) of the demand with the number of scale events (A_u or A_d) for allocated resources. Thus, they compute $Scale$ as $\frac{|du-au|}{du}$, in which the in-

tent is to obtain a value close to 0. In addition, they claim that efficiency is defined by $E = \frac{1}{(A.U)}$, in which A corresponds to the average speed of scaling up, and U refers to the average amount of under-provisioned resources during an under-provisioned period.

Islam et al. [3] provide a quantitative definition of elasticity using financial terms, adopting the point of view of an elastic system customer who wants to measure the elasticity provided by the system. The authors compute the financial penalty for systems' under-provisioning (due to SLA violations) and over-provisioning (unnecessary costs) using a reference benchmark suite to characterize system elasticity. They develop a model to measure a penalty to a consumer for under-provisioning (leading to unacceptable latency or unmet demand) or over-provisioning (paying more than necessary for the resources needed to support a workload). Technically, the penalty model computes numerical integration considering resource demand and supply. Finally, to assess sensitivity to CPU usage, Tembey et al. [10] use the following metric: $\frac{CPUutilization}{allocatedCPU}$.

3. Metrics for HPC Applications in the Cloud

In our previous work, we developed a cloud elasticity model that operates at the PaaS level of a cloud, called AutoElastic [8]. Using the AutoElastic's infrastructure, we propose metrics for HPC applications in the cloud to redefine the so-called speedup and parallel efficiency metrics. Traditionally, HPC applications are executed on clusters or even in grid architectures, in which both have a fixed number of resources. Conversely, cloud elasticity abstracts the infrastructure configuration and technical details about resource scheduling from users, who pay for resources, and consequently energy, in accordance with applications' demands. However, the traditional performance analysis of static infrastructures does not fit elastic environments. Aiming at addressing this gap, the sections 3.1 and 3.2 present new metrics to evaluate elastic cloud performance.

3.1. Elastic Speedup Model

Speedup (S) can be defined for two different types of values, throughput and latency, being commonly defined by the division of M_{old} by M_{new}. M here refers to the value of a metric, and *old* and *new* represent the execution without and with the improvement, respectively. Specifically, one of the most common measurements in computer architecture - the execution time of a program - can be considered a latency quantity. Thus, Equation 1 presents the speedup S as a function of the number of processors p, such that *t(1)* and *t(p)* express the sequential and the parallel processing times. Notice that speedup is a unit-less quantity (the units cancel). Equation 2 denotes the efficiency of a parallel system, describing the fraction of the time that is being used by processors p for a given computation. Efficiency is occasionally preferred over speedup because the former represents an easy means to observe how well parallelization is working.

$$S(p) = \frac{t(1)}{t(p)} \tag{1}$$

$$E(p) = \frac{S(p)}{p} \tag{2}$$

Aiming at observing the possible gain with cloud elasticity for HPC applications, we propose an extension of the previously mentioned speedup and efficiency using these term definitions: elastic speedup (ES) and elastic efficiency (EE). Both ES and EE are explored here in accordance with horizontal elasticity, in which the instances can scale either out or in with the application demands. Furthermore, we consider a homogeneous system in which each VM runs in 100% of a CPU core, regardless of the cloud level. Elastic speedup is calculated by the function *ES(n,l,u)* according to Equation 3, where n denotes the initial number of VMs and l and u represent the lower and upper bounds for the quantity of VMs, respectively. t_{ne} and t_e refer to the conclusion times of an HPC application that was executed without and with elasticity support, respectively. Here, t_{ne} is obtained with the lowest possible number of VMs, in our case l, providing an analogy with sequential execution in the standard speedup.

$$ES(n, l, u) = \frac{t_{ne}(l)}{t_e(n, l, u)} \tag{3}$$

where $l \leq n \leq u$.

3.2. Elastic Efficiency Model

Function *EE(n,l,u)* in Equation 4 computes elastic efficiency. Its parameters are the same as the preceding function ES. Efficiency represents how effective the use of resources is, being the entity positioned last as denominator in the formula. Unlike Equation 2, the resources here are malleable; therefore, we designed a mechanism to reach a single value for them coherently. To accomplish this, EE assumes the execution of a monitoring system that captures the time spent on each configuration of VMs. Equation 5 indicates use of the resources, where $pt_e(i)$ is the application's partial time when running over i VMs. Thus, if elasticity actions do not occur, $pt_e(i)$ and $t_e(n, l, u)$ values will be identical for $i = n$. Equation 4 presents parameter n in the

numerator, which is multiplying the elastic speedup. More precisely, this occurs because $ES(n, n, n)$ is always equal to 1 and $Resource(n, n, n)$ equal to n, so n in the numerator returns an elastic efficiency of 100%.

$$EE(n, l, u) = \frac{ES(n, l, u) \times n}{Resource(n, l, u)} \quad (4)$$

where

$$Resource(n, l, u) = \sum_{i=l}^{u} (i \times \frac{pt_e(i)}{t_e(n, l, u)}) \quad (5)$$

The denominator in Equation 4 was added to capture each infrastructure configuration (from l to u VMs) and its participation in the entire execution, presenting the sum of the partial values as the final value for $Resource(n, l, u)$. The use of a flexible number of resources helps to provide better parallel efficiency when comparing elastic and non-elastic parallel executions. At a glance, there are two approaches to make viable the aforementioned statement: (i) the elastic execution presents an equal, or even a bit greater, execution time when compared with a non-elastic configuration, but we employ a smaller set of resources to reach the first case. (ii) Even devoting a larger number of resources because the CPU-bound HPC application demands them, the gain in the time perspective outperforms the cost. Although scaling in is a key operation to accomplish (i), scaling out is essential for (ii).

4. Evaluation Methodology and Results

The application used in the tests computes the numerical integration of a function *f(x)* in a closed interval $[a, b]$. We used the Composite Trapezoidal rule from a Newton-Cotes postulation [1] to implement a Java application using Sockets. Equation $\int_a^b f(x)\,dx \approx \frac{h}{2}[f(x_0) + f(x_s) + 2.\sum_{i=1}^{s-1} f(x_i)]$ shows the development of the integral in accordance with the Newton-Cotes postulation. The tests were executed in a private OpenNebula cloud, using different combinations of upper and lower thresholds, in the AutoElastic's [8] infrastructure. To evaluate different application behavior, four patterns were modeled: Ascending, Constant, Descending and Wave. All executions started with 2 physical nodes, 4 VMs with a slave process each and a VM running the master process of the application.

Figure 1 presents the evaluated measures considering different threshold settings and load patterns. In the executions with the highest upper threshold 90%, any elastic operation were necessary and, thus, the application used the same resources. Considering all load patterns, an upper threshold equal to 70% was primarily responsible for proving better results in terms of ES index. This threshold implies a more reactive system because the average CPU load of the system will not exceed this limit. Considering that the tested application is CPU intensive, VM allocations happen more often with this threshold. This procedure triggers load balancing among a larger number of slave processes and, consequently, due to the computation grain that remains viable, implies a reduction in the application time. Conversely, this combination of this threshold over the Ascending, Constant and Wave patterns follows the traditional statement of parallel computing; that is, there is a curve along which a greater number of resources entails a better speedup index, but efficiency decreases with time. In other words, this context of load pattern and threshold resulted in a lower EE and in a greater resource consumption.

In contrast to the elastic execution over the Ascending and Constant functions, in which the performance was dictated by the upper threshold of 70%, the behavior of the Descending and Wave patterns is also influenced by the lower thresholds. Specifically, the value of 30% for the lower threshold was responsible for maintaining the allocated resources for longer when compared with 50%; thus, 30% led to a better ES indexes. Exploring the same idea discussed earlier, the computation grain is sufficiently large to direct us to obtain better performance results when a large number of results is considered. Conversely, EE provides the worst results when testing the application with the Descending and Wave functions with the threshold equal to 30%. This happens because the resources are, on average, sparingly used, *i.e.*, they no longer approach close to 100% of utilization. Thus, the employment of 50% for the lower threshold allows reaching better values for EE because the resources are deallocated sooner.

5. Conclusion

This article focused on performance and efficiency measurement of elastic HPC applications on the cloud using the AutoElastic model as a substrate to discuss novel ideas. The scientific contribution of this article is the definition of the new elastic cloud performance metrics: (i) Elastic Speedup (ES); and (ii) Elastic Efficiency (EE). Considering the initial number of VMs and the lower and upper bounds, the article's contribution explores the traditional speedup and efficiency for parallel systems, now considered in elastic infrastructures. Although ES provides possible gains with on-the-fly resource reorganization, EE indicates the effectiveness of resource use. In other words, it would not be interesting to reach a significant ES rate, but pay an EE close to 0. One can use ES and EE to an-

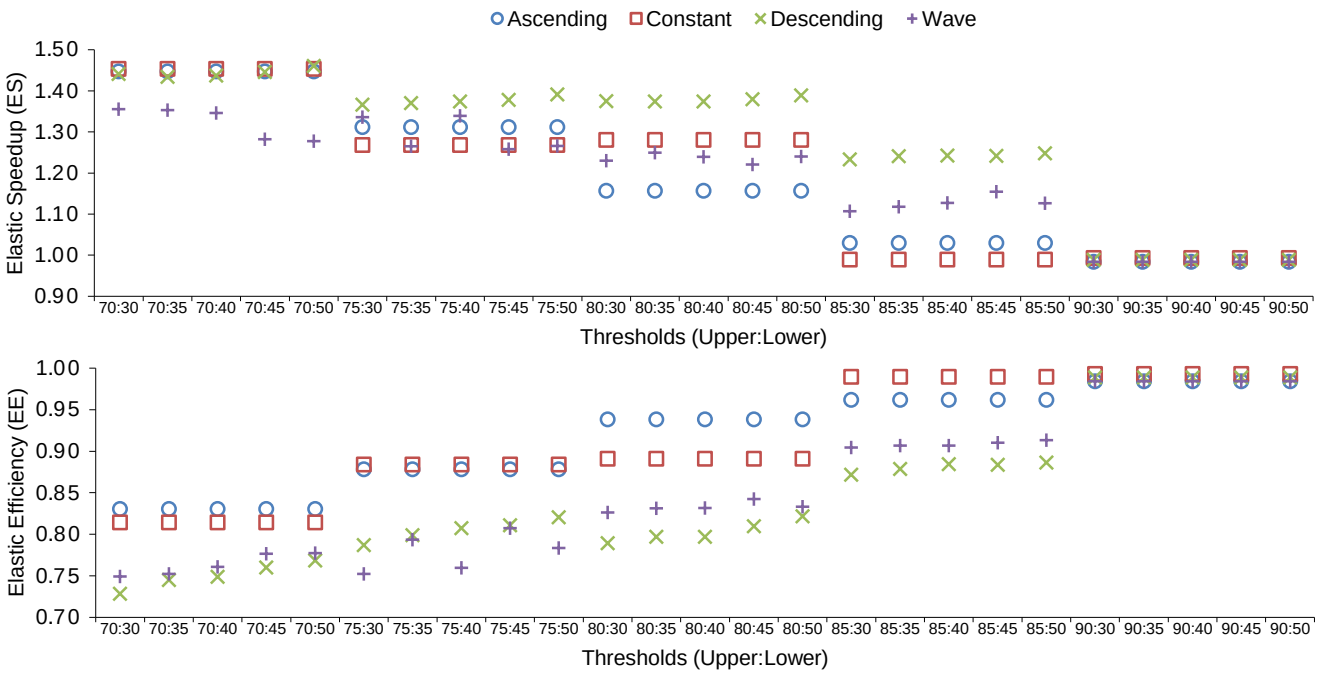


Figure 1. Performance and efficiency results of all application loads in the different scenarios.

alyze algorithms either theoretically, using asymptotic run-time complexity, or in practice, to measure the effectiveness of the use of new available resources.

In future work we intend to explore further the EE and ES concept with different approaches of elasticity.

References

[1] M. Comanescu. Implementation of time-varying observers used in direct field orientation of motor drives by trapezoidal integration. In *Power Electronics, Machines and Drives (PEMD 2012), 6th IET Int. Conf. on*, pages 1–6, 2012.

[2] E. Coutinho, F. de Carvalho Sousa, P. Rego, D. Gomes, and J. de Souza. Elasticity in cloud computing: a survey. *annals of telecommunications - annales des tlcommunications*, pages 1–21, 2014.

[3] S. Islam, K. Lee, A. Fekete, and A. Liu. How a consumer can measure elasticity for cloud platforms. In *Proc. of the 3rd ACM/SPEC Int. Conf. on Performance Engineering*, ICPE '12, pages 85–96, New York, NY, USA, 2012. ACM.

[4] P. Jamshidi, A. Ahmad, and C. Pahl. Autonomic resource provisioning for cloud-based software. In *Proc. of the 9th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 95–104, New York, NY, USA, 2014. ACM.

[5] T. Lorido-Botran, J. Miguel-Alonso, and J. Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592, 2014.

[6] P. Martin, A. Brown, W. Powley, and J. L. Vazquez-Poletti. Autonomic management of elastic services in the cloud. In *Proc. of the 2011 IEEE Symp. on Computers and Communications*, ISCC '11, pages 135–140, Washington, DC, USA, 2011. IEEE Computer Society.

[7] A. Raveendran, T. Bicer, and G. Agrawal. A framework for elastic execution of existing mpi programs. In *Proc. of the 2011 IEEE Int. Symp. on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 940–947, Washington, DC, USA, 2011. IEEE Computer Society.

[8] R. Righi, V. Rodrigues, C. Andre daCosta, G. Galante, L. Bona, and T. Ferreto. Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *Cloud Computing, IEEE Transactions on*, PP(99):1–1, 2015.

[9] E. Roloff, M. Diener, A. Carissimi, and P. Navaux. High performance computing in the cloud: Deployment, performance and cost efficiency. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th Int. Conf. on*, pages 371–378, Dec 2012.

[10] P. Tembey, A. Gavrilovska, and K. Schwan. Merlin: Application- and platform-aware resource allocation in consolidated server systems. In *Proc. of the ACM Symp. on Cloud Computing*, SOCC '14, pages 14:1–14:14, New York, NY, USA, 2014. ACM.

[11] A. Weber, N. R. Herbst, H. Groenda, and S. Kounev. Towards a Resource Elasticity Benchmark for Cloud Environments. In *Proc. of the 2nd Int. Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS 2014), co-located with the 5th ACM/SPEC Int. Conf. on Performance Engineering (ICPE 2014)*. ACM, March 2014.

Improvement of a Parallel File System Simulator by Data Striping Implementation

Vivien Michel
Polytech'Grenoble
Saint-Martin-d'Hères, France
Vivien.Michel@e.ujf-grenoble.fr

Francieli Zanon Boito
Instituto de Informática
Universidade Federal do Rio Grande do Sul, Brazil
fzboito@inf.ufrgs.br

Mario A. R. Dantas
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina, Brazil
mario.dantas@ufsc.br

Abstract

In this paper, we discuss modifications to the PFSSim parallel file system simulator aiming to improve its accuracy. Simulation is an important tool for research as it allows for experimentation in situations where a real system is not available. However, accuracy is essential for simulators' usefulness. PFSSim has an accuracy problem due to not implementing data striping. We have implemented this operation and evaluated the new version of PFSSim against a real system. Our results show good results with the file-per-process approach and small requests, but also highlight the importance of adequate disk access simulation.

1. Introduction

High performance computing (HPC) applications rely on parallel file systems (PFS) to achieve good performance when handling large amounts of data. These systems distribute files' data - though an operation called *data striping* - among multiple data servers, which can be accessed in parallel for performance. Because of the historic gap between processing and data access speeds, there is an active research field aiming to improve parallel I/O techniques.

Research on parallel file systems requires experiments on large scale architectures in order to understand how the interaction between the different levels of the I/O stack affects performance behavior. Such experiments are also needed to validate new techniques, test predictive models' accuracy, etc. Nonetheless, researchers often do not have access to supercomputers or are not allowed to allocate the machine's time for their experiments. Even when access is possible, the experiments may require deep modifications in the system, which may not be allowed. In this context, a simulator is an important tool for research. Through a simulator, researchers are able to run needed experiments and explore deep changes that would not be possible in a real system. Moreover, using a simulator saves money in supercomputer time and energy consumption.

Simulation is an important and powerful tool as long as results are accurate. In other words, for a simulator to be useful, it needs to correctly represent the performance and behavior of a real system. In this paper, we discuss modifications we have implemented in the PFSSim [3, 4] parallel file system simulator for correctly simulating data striping and its effects. These modifications were performed aiming to improve the simulator's accuracy, specially for experiments with I/O scheduling algorithms.

The rest of this paper is organized as follows: the next section briefly presents the PFSSim simulator. Section 3 discusses the data striping implementation. Results obtained with both versions of the simulator (before and after our modifications), compared with results obtained from a real system, are discussed in Section 4. Section 5 brings final remarks and future work.

2. The PFSSim simulator

PFSSim is a parallel file system simulator built over the Omnet++ [8] network simulator. The file system components (client, data server, metadata server, and I/O gateway) and local components (local file system, virtual file system layer, disk cache, and disk) were implemented through the Omnet++ infrastructure. When configuring a simulation, the user must model the network topology and connect the components through the NED description language.

The system behavior was meant to replicate the Parallel Virtual File System (PVFS) [6] version 2.8.2, but without its implementation specifics. Therefore, the authors claim other file systems, such as Lustre [1], can also be simulated by adjusting some parameters.

An interface to describe I/O scheduling algorithms - to work in the context of requests to PFS servers - is offered by the simulator through the I/O gateway component (called "proxy"), which can be placed at client-side or server-side (by connecting it to the client or server component through a perfect channel). There is also the possibility of placing these components as intermediate machines between clients and servers, replicating an I/O forwarding organization. There is a communication channel between proxies that can be used to coordinate I/O scheduling instances.

Other parallel file system simulators [5, 2] have been proposed in the literature. However, we have chosen to work with PFSSim due to its capabilities for I/O scheduling algorithms simulation.

A weakness of PFSSim is that it does not implement the data striping operation, where a file is striped into fixed size portions and these portions are distributed among the data servers following a round-robin approach. In its original implementation, each data server receives a "disk image" where the whole file is present.

Since I/O scheduling algorithms work to adjust the access pattern at the servers, the access to the storage device is an important factor in the resulting performance. This access time is directly affected by the offset distance between consecutive accesses. For these reasons, the simulator's accuracy is affected due to the disk image not reflecting what would be observed in a real system. The next section will describe the modifications we have done in the simulator in order to adequately implement data striping.

3. Modifications in the PFSSim simulator

The PFSSim implementation includes some scripts to generate input files required for the simulation. One of them is in charge of creating a file for each storage device. Before the modification, the size of the whole files were used to write this configuration file, and all servers received identical files. Next, the configuration file is read by the simulated local file system. In other words, the file system on each disk have the whole files inside. The improvement consists in the distribution of the total size of each file on each server based on the "stripe size". The algorithm can be splitted in three parts:

1. **Use the previously generated layout file**: the first step consists in parsing the layout file in order to know how files are striped. In fact, for each simulated file, the metadata server knows the stripe size and the number of the n_{th} server, since the first server (the one receiving the first portion of the file) is not necessarily server zero. With this parsing, the script takes all information needed to split the given amount of data among the servers. Additionally, the stripe size unit can be kilobytes or megabytes.
2. **Split data among the servers**: then data must be split among the servers accordingly to previously recovered information. Each server has its own total amount of data which is a subdivision of the total file size.
3. **Write data on per-server files**: finally, each file and server have their own amount of data (in bytes) to write in an individual "disk image" file. They will be read by the simulator at its initialization.

In order to verify if these changes were enough to correctly implement data striping, it was necessary to verify if offsets requested by clients were relative to the data servers' local files or to the global file stored in the PFS. Through a script that logs non-contiguous accesses to data servers, we have confirmed that offset were already correctly translated.

4. Performance Evaluation

We have conducted a performance evaluation in order to see how accurate simulation results are before and after our modifications. For that, we have compared the simulated results to those obtained from a real system. The next section describes the evaluation methodology, and Section 4.2 discusses the obtained results.

4.1. Methodology

The real system used in our experiments was the Graphene cluster from *Grid'5000*. Graphene's nodes have a quad-core Intel Xeon $X3440$ 2.53GHz, 16GB of RAM and a 320GB SATA II hard disk. The operating system used was Debian 6 (Squeeze) - kernel 2.6 - with MPICH2, PVFS version 2.8.2 (not the most recent version, but the same considered during the simulator's development).

We have used 1 node as a PVFS metadata server, 6 nodes as data servers and 32 as clients. The used stripe size was 64KB. Tests were executed on the clients through the IOR[1] benchmarking tool (version 3.0.1), which allows access patterns description. We have executed tests varying the following aspects:

- file-per-process approach or shared file (independent files have 2GB each, the shared file has 2GB per client);

1 https://github.com/chaos/ior

- request size: 32KB (smaller than the stripe size) or 1MB (larger than the stripe size times the number of servers);
- read or write tests;

The Graphene's Gigabit Ethernet interconnection was modeled in Omnet++ and its nodes' parameters (such as page size, dirty_ratio, etc) were obtained to configure the simulations. In the simulator, proxy nodes were placed with the data servers (one proxy per server) and we have experimented with the EDF, SFQ and FIFO scheduling algorithms. Nonetheless, initial tests have not presented significant differences between results with different scheduling algorithms, so we have proceeded with the tests using the EDF algorithm only. The LMBench [7] benchmark was used to measure the time needed to read data from the main memory - the buffer cache - in Graphene's nodes and we have used this value to configure the simulator.

The presented results correspond to the arithmetic average of 5 repetitions in order to reduce the influence of aleatory aspect.

4.2. Results

All tested combinations between the parameters listed in the previous section added up to 14 experiments. Due to space constraints, we have chosen to show here 3 of these tests we consider to be representative of the observed behaviors.

Figure 1 presents these three sets of results. All of them are for read tests. Figures 1a and 1b show results obtained with the file-per-process approach with different request sizes - 1MB and 32KB. The results from tests where all processes share a file (with 1MB requests) are presented in Figure 1c. Simulation results are represented by the first (before the data striping implementation) and second (after the data striping implementation) bars. The third bars represent the results observed in the real system. The y axis gives the total I/O time in seconds. The times to open and close the file are not included in the results.

For the tests where each client has an independent file, the correct data striping results in shorter I/O times. This happens because files are stored sequentially in the disk image. When an access to file B is done right after an access to file A, the offset distance used to obtain the disk access time depends on A's size (and size of all files stored between A and B, if this is the case). With data striping, each file is smaller on each data server's storage device (because each data server has only a part of this file) and hence offset distances are smaller. PFSSim considers disk access time to be linearly proportional to the offset distance, so smaller offset distances result in smaller disk access times, decreasing the simulated time.

Our best results - where the new version of the simulator accurately represents the real system - were observed with the file-per-process approach with small requests (Figure 1b). As the request size increases (Figure 1a), the observed error increases. We believe this is due to PFSSim's disk access time modeling. Results obtained by UFRGS' researchers working in the SeRRa project[2] suggest that considering access time to be linearly proportional to the offset distance **does not** reflect real disk behaviors for large requests.

On the other hand, in tests with the shared file approach (Figure 1c), there were no significant difference between times obtained by both versions of the simulator. This happened because most accesses were served by the buffer cache - in main memory - so disk access and data striping played no role. At each server, after the first access to the shared file, **prefetching** is able to load data to serve future requests to the main memory. We can see that PFSSim underestimates I/O time in this situation. We believe this happens because the simulated prefetching mechanism is unrealistically successful in masking disk access time. This is supported by the results obtained for the real system: in all three tests the same total amount of data is accessed from the file system, but I/O time in Figure 1c is the highest. If all accesses were served by the buffer cache, it should be the lowest.

5. Conclusions

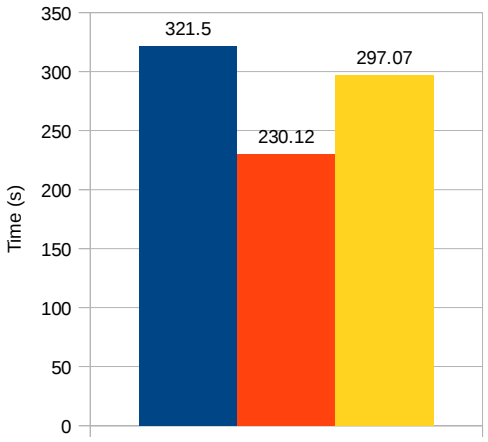
This paper discussed an improvement to a parallel file system simulator called PFSSim. It was chosen due to its I/O scheduling simulation capabilities, which are in the research group's interests. We have implemented the data striping operation aiming to improve disk access simulation.

We have evaluated the two versions of the simulator against a real system. Good results have been observed for the file-per-process approach with small requests. Nonetheless, other tested situations presented poor results. Ongoing work focuses on improving disk access simulation by using models provided by the SeRRa devices profiling tool.

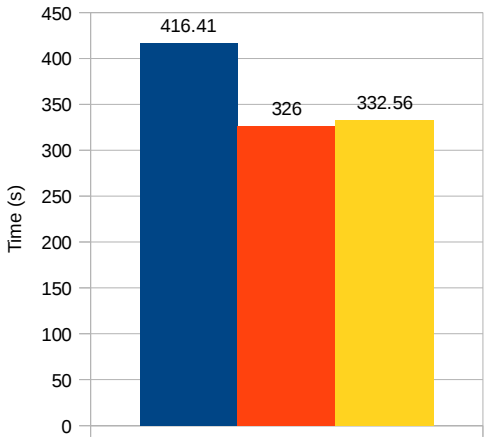
6. Acknowledgements

The experiments presented in this paper were carried out on the Grid'5000 experimental test bed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

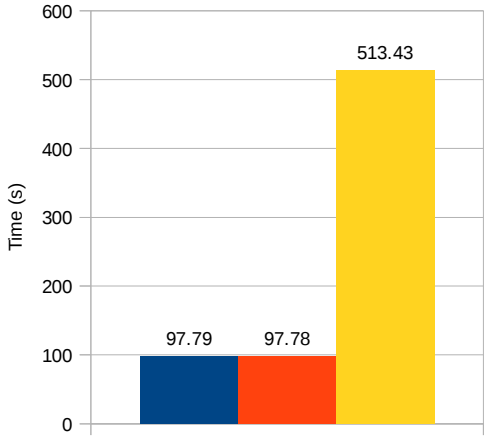
2 http://serratool.bitbucket.org



(a) File-per-process - 1MB requests



(b) File-per-process - 32KB requests



(c) Shared file - 1MB requests

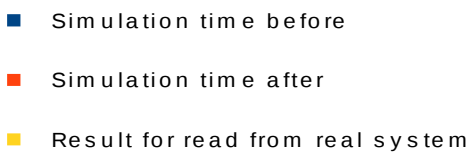


Figure 1: Results comparing two versions of the simulator with a real system

References

[1] Lustre: A scalable, high-performance file system, November 2002. http://www.lustre.org/docs/whitepaper.pdf.

[2] P. H. Carns, B. W. Settlemyer, and W. B. Ligon, III. Using server-to-server communication in parallel file systems to simplify consistency and improve performance. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 6:1–6:8, Piscataway, NJ, USA, 2008. IEEE Press.

[3] Y. Liu, R. Figueiredo, D. Clavijo, Y. Xu, and M. Zhao. Towards simulation of parallel file system scheduling algorithms with pfssim. In *Proceedings of the 7th IEEE International Workshop on Storage Network Architectures and Parallel I/O*, 2011.

[4] Y. Liu, R. Figueiredo, Y. Xu, and M. Zhao. On the design and implementation of a simulator for parallel file system research. In *Mass Storage Systems and Technologies (MSST), IEEE 29th Symposium on*, pages 1–5, May 2013.

[5] E. Molina-Estolano, C. Maltzahn, J. Bent, and S. Brandt. Building a parallel file system simulator. In *Journal of Physics: Conference Series*, volume 180, page 012050. IOP Publishing, 2009.

[6] R. B. Ross, R. Thakur, et al. Pvfs: A parallel file system for linux clusters. In *Proceedings of the 4th annual Linux Showcase and Conference*, pages 391–430, 2000.

[7] C. Staelin. lmbench: Portable Tools for Performance Analysis. In *In USENIX Annual Technical Conference*, 1996.

[8] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, pages 60:1–60:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

Faster Storage Devices Profiling with Parallel SeRRa

Jean Luca Bez, Francieli Zanon Boito, Rodrigo Virote Kassick,
Vinicius Rodrigues Machado, Philippe O. A. Navaux
Institute of Informatics – Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
{jlbez, fzboito, rvkassick, vrmachado, navaux}@inf.ufrgs.br

Abstract

This work presents the parallel storage device profiling tool SeRRa. Our tool obtains the sequential to random throughput ratio for reads and writes of different sizes on storage devices. In order to provide this information efficiently, SeRRa employs benchmarks to obtain the values for only a subset of the parameter space and estimates the remaining values through linear models. The MPI parallelization of SeRRa presented in this paper allows for faster profiling. Our results show that our parallel SeRRa provides profiles up to 8.7 *times faster than the sequential implementation, up to* 895 *times faster than the originally required time (without SeRRa).*

1. Introduction

Hard Disk Drives (HDDs) have been the main non-volatile storage devices. For this reason, most systems were developed or adapted in order to maximize performance when accessing these devices [7].

Solid State Drives (SSDs) are a recent alternative to hard disks. These devices use flash memory to store data, which brings advantages [3]. Nonetheless, despite the growing adoption of SSDs, their larger cost per byte still hampers their use on large-scale systems for HPC. Therefore, several parallel file system deployments on clusters still store data on hard disks. Furthermore, hybrid solutions where both devices are used have been gaining popularity [6].

Another popular solution for storage on HPC systems is the use of RAID arrays that combine multiple hard disks onto a virtual unit for performance and reliability purposes. Data is striped among the disks and can be retrieved in parallel, which improves performance.

Several assumptions about performance from HDDs do not hold when using SSDs and RAID arrays, and different requirements arise [5]. Therefore, we cannot simply classify optimizations by saying they are only suitable for HDDs or SSDs. Approaches that aim at generating contiguous accesses (originally designed for HDDs) can greatly improve performance when used on SSDs that are also sensitive to access sequentiality. Furthermore, on any device, the performance improvement caused by the use of a specific optimization may not compensate its overhead. Hence, these optimizations could be classified according to the *sequential to random throughput ratio* that devices must present in order to benefit from them.

However, obtaining this metric to a storage device can be a time-consuming task. In order to provide accurate results as fast as possible, SeRRa was developed in our previous work [1]. In this scenario, this paper proposes a parallel implementation of SeRRa. We evaluate our approach over clusters with HDDs, SSDs, and RAID arrays, and show performance improvements over the sequential version of the tool.

This paper is organized as follows: the next section presents the design of SeRRa. Section 3 details the parallel implementation. Section 4 discusses results obtained with the new SeRRa. Finally, Section 5 brings final remarks.

2. SeRRa: A Storage Device Profiling Tool

This section describes SeRRa, a storage device profiling tool. Its development was motivated by the need for a fast way to obtain the sequential to random throughput ratio from storage devices. The main goals of SeRRa's project are performance - the information must be provided as quickly as possible - accuracy, and generality.

Keeping both performance and accuracy goals at the same time is a challenging problem because profiling a storage device adequately can take a long time. Table 1 presents the time spent to profile the devices used in this study. It includes time required for one execution of the tests and for the complete profiling (that includes several executions in order to provide statistical guarantees). Details about these devices and the executed tests will be presented in Section 4. From the times reported in Table 1, it is clear that these tests

	One execution	Total profiling
Pastel (HDD)	15.22	329.49
Graphene (HDD)	12.26	120.38
Suno (RAID-0)	4.21	61.32
Edel (SSD)	3.09	40.44

Table 1: Original time in hours to profile the storage devices used in this study.

are time consuming, as the fastest profile took over 1.5 days, while the slowest one took almost 14 days.

The slow profiling whose times are presented in Table 1 consists of executing an I/O benchmark several times varying file sizes, request sizes and operation (sequential read, random read, sequential write, and random write). We have observed that most of the access times graphs (time to process a request as a function of the request size) present a linear function appearance. Because of this observation, we have decided to use linear regressions on the design of our profiling tool. Therefore, the following steps compose SeRRa's execution:

1. **Monte Carlo:** request sizes inside a given interval are randomly picked.
2. **Benchmark:** the test is executed for the request sizes picked on the previous step. For this task, SeRRa uses the $IOzone$ benchmark [4].
3. **Linear Regression:** the complete set of access time, for each given interval, is estimated through linear regression based on the measurements from the previous step. Each test (read or write, sequential or random) for each interval is estimated separately.
4. **Report:** The sequential to random throughput ratios for the read and write tests are reported. The tool provides such values for all request sizes, as well as averages, maximum and minimum values. The estimated access times curves are also provided.

The tool, implemented on *Python*, is open source and available at `http://serratool.bitbucket.org/`.

3. Parallel SeRRa

Although SeRRa profiles a machine's storage device through its local file system, it can also be used in the context of a Parallel File System (PFS). The information provided by the tool can be used by I/O optimization techniques to improve performance when accessing those systems.

Therefore, optimization techniques can benefit from knowing how the PFS data servers' storage devices behave regarding access sequentiality. This information can be used, for instance, to decide between different I/O scheduling algorithms.

It is usual for HPC architectures to dedicate a set of nodes for the parallel file system deployment. This shared storage infrastructure is often homogeneous, with identical storage devices on all involved machines. These devices are expected to present the same performance behavior, and thus the same sequential to random throughput ratios.

For this reason, we have decided to create a parallel implementation of SeRRa which benefits from this characteristic to provide information faster. A faster profiling facilitates the tool's use for dynamic decision making.

SeRRa's parallel implementation, developed with MPI4PY[1], uses the master-slave paradigm for communication between processors. We have parallelized the step of benchmarking (step 2 described in the previous section), since it is the most time-consuming one. The master is responsible for all other steps, such as picking measuring points, linear regressions and reporting results. Additionally, the master sends tasks to slaves and receives their results until there are no tasks left. Each task corresponds to a request size to be tested with four access patterns: sequential write, random write, sequential read and random read.

Therefore, it makes no sense to use more processes than available machines, since this approach would lead to benchmarks being executed concurrently in the same node, compromising results. Furthermore, in this implementation the parallelism is limited by the number of intervals, measuring points per interval and benchmark repetitions.

4. Performance Evaluation

The four systems used in our tests are listed in Table 2. They were selected by their variety, aiming at representing most available devices. All of them are homogeneous clusters from *Grid'5000* [2].

The tests were executed on the *Linux* operating system, using *IOzone* version *3.397*. All tested devices were accessed through the *Ext3* local file system, and the default *cfq* disk scheduler was kept. Both virtual memory's page size and file system's block size are 4KB. Caching was explicitly disabled through the *O_DIRECT* POSIX flag (*"-I"* parameter for IOzone).

Five different file sizes were used - 40MB, 200MB, 400MB, 800MB and 1200MB. On all tested devices, we observed that the access time curves usually stabilize before 1200MB, not showing significant differences as we increase the file size further. The request sizes lied within two ranges: from 8KB to 64KB with gaps of 8KB; and from 64KB to 4MB with gaps of 32KB.

1 http://packages.python.org/mpi4py

Cluster	RAM (GB)	Storage Device		
		Type	Capacity	Bus
Pastel	8	HDD	$250GB$	SATA $1.5Gb/s$
Graphene	16	HDD	$320GB$	SATA $3Gb/s$
Suno	32	RAID-0	$2 \times 300GB$	SAS $3Gb/s$
Edel	24	SSD	$64GB$	SATA $1.5Gb/s$

Table 2: Configuration of the evaluated storage devices

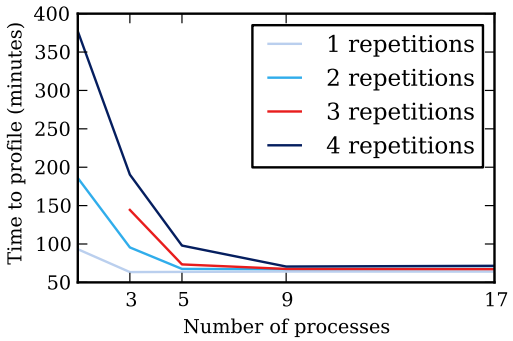
We have executed tests using two measuring points per interval, as this configuration was the one with the best results in our previous analysis [1]. We use up to four benchmark repetitions. Statistically, executing more repetitions of the benchmarks is expected to provide better results.

We compare the parallel implementation's results with SeRRa's sequential implementation and with the profile obtained without the tool. To obtain the latter, we have executed the complete profiling - without estimations, executing the benchmarks for all the specified request sizes - on all tests environments with the aforementioned parameters. All tests were repeated until a 90% confidence could be achieved with a *t-student* distribution - with at least six executions. The maximum accepted error was of 10%. The necessary time to obtain this complete profile per machine, up to 14 days, was previously presented in Table 1.

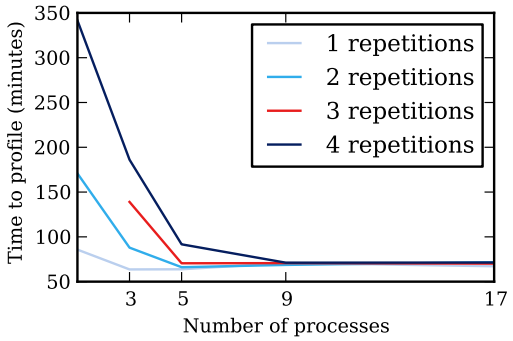
Table 3 presents the total time to perform all experiments described in the previous section by the different implementations. The numbers between parenthesis represent fractions of the originally required time (without SeRRa). The results with parallel SeRRa from the table were obtained using 17 processes (a master and 16 slaves). We show the times for sequential SeRRa with 1 and 4 benchmark repetitions. Repeating the benchmarks multiple times and taking the results' average allows for more accurate results. On the other hand, the time required to obtain a profile increases linearly with the number of repetitions.

We can see the parallel version's times for 4 benchmark repetitions were faster than (but close to) sequential SeRRa with 1 repetition. In other words, it is possible to obtain more accurate results using roughly the same profiling time.

Figure 1 presents the profiling time with SeRRa on the Pastel and Graphene clusters varying the number of benchmark repetitions. Graphs for Suno and Edel are similar to Graphene's and were omitted due to space constraints. The x axis represents the number of processes. The single-process case (the first point of each line) is the time obtained with the sequential implementation. Onward, the number of processes depicted includes the master plus the slaves – i.e.



(a) Pastel cluster (HDDs)



(b) Graphene cluster (HDDs)

Figure 1: Time to profile disks with Parallel SeRRa

the number of processes that actually run benchmarks is the number of processes minus 1.

We can see that performance increases (the profiling time decreases) as we increase the number of processes. With few repetitions, performance reaches its best with few processes and does not profit from more nodes to run the profiling. This happens due to the parallelism limit of our experiment: using two intervals with two measuring points per interval, the total number of tasks is $2\ intervals\ \times\ 2\ points\ per\ interval\ \times\ N\ repetitions$. Therefore, using 2 benchmark repetitions we were expected to decrease the profiling time until 8 slaves (9 processes), and with 4 repetitions until 17 processes.

However, we can see that in most cases (in Pastel, Graphene, and Suno) there was no difference between the results with 4 benchmark repetitions using 9 or 17 processes. The same happened for results with 2 repetitions considering 5 or 9 processes. This indicates that the maximum speedup can be reached using a number of tasks which is twice the number of slave processes. This configuration leads to a better load balance, as it will be discussed later in this section.

Table 4 presents the observed speedups, comparing the best result from the parallel implementation with the sequential one. The speedup is not expected to be linear with

Cluster	No SeRRa	Sequential SeRRa		Parallel SeRRa
		1 repetition	4 repetitions	4 repetitions
Pastel (HDD)	19769.4	93.21 (1/212)	376.99 (1/52)	71.57 (1/276)
Graphene (HDD)	7222.8	85.69 (1/84)	341.50 (1/21)	70.75 (1/102)
Suno (RAID-0)	3679.2	28.12 (1/130)	109.42 (1/33)	21.56 (1/171)
Edel (SSD)	2426.4	5.92 (1/409)	23.58 (1/102)	2.72 (1/892)
Sum	33097.8	212.94 (1/155)	851.49 (1/39)	166.6 (1/199)

Table 3: Time to profile in minutes. The fractions compare SeRRa's results with the originally required time (without SeRRa).

the number of processes, since the tasks distributed to the nodes are of different sizes: considering a fixed file size (as SeRRa does), it takes longer to execute the tests with smaller request sizes. In other words, the parallel implementation profiling time will be limited by the slowest test.

Benchmark repetitions	**Pastel (HDD)**	**Graphene (HDD)**	**Suno (RAID-0)**	**Edel (SSD)**
1	1.47	1.37	1.31	2.29
2	2.76	2.59	2.6	4.51
3	2.15	1.98	2.18	3.31
4	5.34	4.83	5.08	8.71

Table 4: Speedup provided by SeRRa's parallel implementation (with the best number of processes to each case).

We can observe from Table 4 that speedups observed for the Edel cluster are the highest. This happens because Edel's devices present the lowest sequential to random throughput ratios. When storage devices present a high sequential to random throughput ratio, a task which has to perform many small, random accesses to access a fixed-size file will take much longer than a task which accesses a file of the same size with larger, random or sequential requests. This will create a situation of task imbalance which can impair speedup due to longer execution times.

The load imbalance explains why the best performance was achieved having a number of processes which is half the number of tasks for Pastel, Graphene, and Suno; and why this did not happen for Edel, where the difference between the tasks is smaller due to lower sequential to random throughput ratios. Having more available tasks than processes allows for better load balancing. Therefore, one possible solution to avoid the imbalance problem would be to break the tests in smaller units.

5. Final Remarks

This paper presented a parallel implementation of a tool for storage devices profiling regarding access sequentiality named SeRRa. Decreasing SeRRa execution time is important because it facilitates its use by I/O optimizations to dynamically adapt to storage devices' characteristics. For this reason, we have developed a parallel implementation with MPI following a master-slave paradigm. This approach is adequate for homogeneous storage nodes.

We have evaluated our approach with four clusters, using HDDs, RAID arrays and SSDs. Our results show performance improvements (decreases in the profiling time) of up to 8.71 times with the parallel implementation of SeRRa over the sequential one, up to 895 times over not using SeRRa.

References

[1] F. Z. Boito, R. V. Kassick, P. O. Navaux, and Y. Denneulin. Towards fast profiling of storage devices regarding access sequentiality. In *Symposium on Applied Computing (SAC)*, Salamanca, Spain, 2015. ACM.

[2] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[3] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *11th International Joint Conference on Measurement and Modeling of Computer Systems*, pages 181–192, New York, NY, USA, 2009. ACM.

[4] W. D. Norcott and D. Capps. *Iozone filesystem benchmark*, 2006. Available at www.iozone.org. Accessed in March 2014.

[5] A. Rajimwale, V. Prabhakaran, and J. D. Davis. Block management in solid-state devices. In *Proceedings of the USENIX Annual Technical Conference*, pages 279–284, San Jose, CA, USA, 2009. USENIX Association.

[6] W. Xu, Y. Lu, Q. Li, E. Zhou, Z. Song, Y. Dong, W. Zhang, D. Wei, X. Zhang, H. Chen, et al. Hybrid hierarchy storage system in milkyway-2 supercomputer. *Frontiers of Computer Science*, 8(3):367–377, 2014.

[7] Y. Zhang and B. Bhargava. Self-learning disk scheduling. *IEEE Transactions on Knowledge and Data Engineering*, 21(1):50–65, 2008.

Comparison of sequential and parallel algorithms for word and context count

Eduardo D. Ferreira[1], Francieli Zanon Boito[2], Aline Villavicencio[1]
[1]Grupo de Processamento de Linguagem Natural
[2]Grupo de Processamento Paralelo e Distribuído
Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, Brasil - {edferreira, fzboito, avillavicencio}@inf.ufrgs.br

Abstract

A distributional thesaurus has a great importance for a series of NLP applications, but its creation takes a long time to be processed. To make this thesaurus creations faster, we have developed a parallel implementation of the word and context count phase, the first step in the counting method based thesaurus creation. Our results have shown a considerable gain in performance provided by the parallelization.

1. Introduction

The automatic creation of distributional thesaurus [6, 1] is of fundamental importance for a series of applications in Natural Language Processing (NLP). However, as this requires very large amounts of data to produce good quality thesaurus, the result is a long processing time, making it sometimes impossible to build the thesaurus [2, 8]. Aiming to make this acquisition faster, we have studied the parallelization of the thesaurus creation. The parallelization aims only to reduce the run time and must produce the same result obtained by the sequential version.

This paper focuses on the parallelization of the phase of word and contexts count for the creation of distributional thesaurus. A comparison is drawn between the times of the sequential bash implementation, from the Minimantics package [7][1] and the parallel implementation developed with Spark[2] that performs operations in parallel using a cluster of computers.

The remainder of this paper is organized as follows: the next section describes the background on distributional thesaurus, Section 3 discusses the word and context count algorithm and its parallel implementation. Results are presented in Section 4 and related work in Section 5, followed by conclusions and future work.

1 Minimantics: A MINImalist multi-threaded tool in C for building standard distributional seMANTICS models. https://github.com/ceramisch/minimantics

2 https://spark.apache.org/

2. Distributional Thesaurus

A thesaurus is a list of words associated by a specific characteristic, such as the similarity between them. Its construction is traditionally manual and results in a high-quality, but with low coverage and high costs.

To solve this problem, the creation of thesaurus can be done automatically from texts, based on the distributional hypothesis of Harris [4]. It assumes that you can find the meaning of a word by the words surrounding it. One of the main approaches in the creation of thesaurus is the counting method [6, 1].

To create a thesaurus using the counting method, initially all occurrences of all words in the text are extracted with the words that surround them within a fixed-size window. Then the total co-occurrence count is made for each pair. With these counts, a matrix is created containing the number of times that each word occurred with each other word in the text, as seen in the example provided by Table 1. After that, for each target word (line), the association between it and its contexts (columns) is calculated. [6, 1].

	Delicious	Eat	Expensive
Chocolate	7	3	5
Pizza	3	9	4

Table 1. Example of a words and contexts matrix

3. Parallelization of the Algorithm

We have implemented a parallel version of the word and contexts counting algorithm. The input to this phase is a list

of word pairs (target and context) and the output is the number of occurrences of each of these pairs in the input. Table 2 presents an example of input. Providing this input to the algorithm would result in the output shown in Table 3.

Target	Context
Chocolate	Eat
Chocolate	Delicious
Chocolate	Expensive
Chocolate	Delicious

Table 2. Algorithm input example

Target	Context	Count
Chocolate	Eat	1
Chocolate	Delicious	2
Chocolate	Expensive	1

Table 3. Algorithm output example

This type of processing fits the MapReduce [5] programming paradigm. That happens because subsets of the input lines can be considered separately for occurrences counting, and then all results can be grouped. In other words, the input text can be divided between multiple tasks (Map), where each task will count its part independently, and finally the counted occurrences of the same pairs will be added (Reduce). We chose to use Spark because it is a suitable tool to handle large amounts of data, providing an environment for developing MapReduce programs. We have used Scala for writing the parallel implementation, because of it is a high-level language with great expressiveness .

3.1. Algorithm

The parallel algorithm (in Scala) used for the word count is as follows. Arguments are input file name (0), minimum frequency to filter (1) and output file name (2).

```
object CountTriples {
  def main(args: Array[String]){
    val textFile = sc.textFile(args(0))
      val counts = textFile.map(word =>
          (word, 1)).reduceByKey(_ + _)
      val count_filter = for((key,value)
          <- counts if (value >=
          (args(1).toDouble))) yield(key,
          value)
      count_filter.saveAsTextFile(args(2))
          } }
```

4. Results

For the experiments presented in this paper, we have used the Spark framework version 1.3.1 and Scala version 2.11.6 in the *Sagittaire* cluster from Grid'5000[3]. Up to 40 nodes from the cluster were used. Each node is equipped with 2 AMD Opteron 250 2.4GHz (single core), 2GB of RAM, and a 73GB hard disk (via SCSI). The nodes are connected through a Gigabit Ethernet network. The Debian 6 ("Squeeze") operating system was used. During the parallel experiments, one node acts as the master, while the others are slaves.

Two subsets of the UKWaC corpus [3] were used: one with 68KB and another with 11GB. The files were copied to all nodes before running the parallel code. The measured time to copy the 11GB file between any pair of nodes from the Sagittaire cluster is approximately 280 seconds. Since spreading a piece of information to a set of N nodes can be achieved in $O(log_2 N)$, we could extrapolate this measurement to consider the required times to get the input file to all nodes presented in Table 4.

10 nodes	20 nodes	40 nodes
929.6 s	1209.6 s	1489.6 s

Table 4. Estimated time to copy the 11GB input file to all nodes involved in the execution (seconds).

In the study presented in this paper, we did not consider the use of a distributed file system such as HDFS [10] or NFS [9]. This will be subject of future work.

The obtained results are shown in Table 5 for the 68KB corpus and in Table 6 for the 11GB input. The times presented are the arithmetic mean of up to 8 executions, and the standard deviations are also presented in the tables. Performance gains were observed only for the large corpus (Table 6): from $\approx$ 14000 to 180 seconds, a gain of 77.57%. These results are also represented in Figures 1 and 2. Is important to notice that each of these nodes runs with two cores.

Comparing the times obtained for the parallel and sequential implementations, speedup and efficiency metrics were calculated. The former represents the performance improvement achieved by the parallel version, while the latter relates this improvement with the amount of resources (cores) used.

We can observe that the speedup achieves a higher value than the number of cores in the results with 10 and 20 machines. This happens because, although both versions im-

3 http://www.grid5000.fr/

	Sequential	Parallel (10 nodes, 20 cores)	Parallel (20 nodes, 40 cores)	Parallel (40 nodes, 80 cores)
Time (s)	14029.8	536.74	289.85	180.87
Standard Deviation	0	1.056	1.46	3.3
Speedup		26.13	48.4	77.56
Efficiency	1	1.3	1.21	0.96

Table 6. Results with the 11GB corpus

	Sequential	Parallel (40 nodes)
Time (s)	0.09	45.31
Standard Deviation	0.00	0.95
Speedup		0.002
Efficiency	1	0.00

Table 5. Results with the 68KB corpus

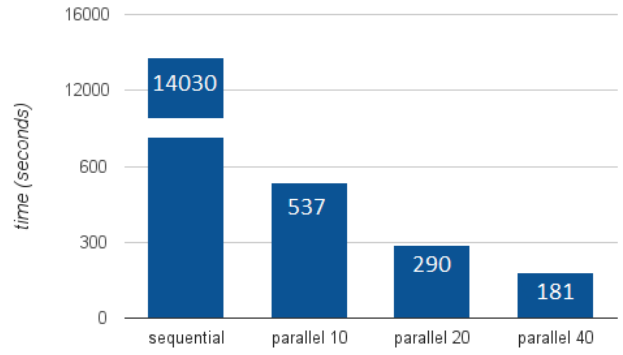


Figure 1. Time with the 11GB corpus

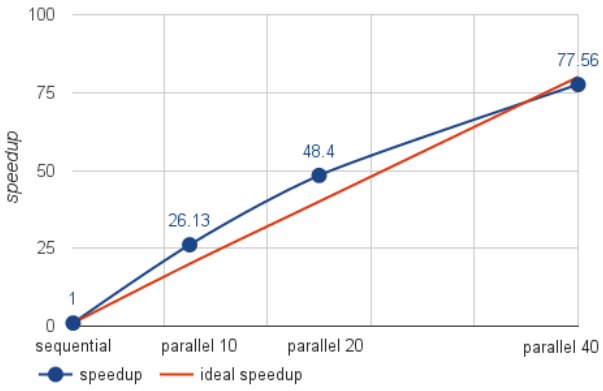


Figure 2. Speedup with the 11GB corpus

plement the same algorithm, the sequential version sorts the pairs before generating the output file, while in the parallel version the sort is not necessary because of how Spark represents data as key-value entries. In the sequential version, the output file must be sorted before going to the next thesaurus creation step. However, in the parallel implementation data can be directly handled by the next stage.

Speedup and efficiency decrease as the number of nodes involved in execution increases. One possible explanation for this is that the ratio between tasks generated and nodes becomes sub-optimal for the tests environment, due to the cost of maintaining a large number of slaves. In this case, greater efficiency could be obtained for 40 processing nodes with a larger input corpus.

Considering the necessary time to transfer the original file to all nodes presented in Table 4 we can calculate the total execution time, presented in Table 7. We can see that speedup is obtained even when considering the time to copy the file to all nodes. Moreover, the best speedup and efficiency were observed with 10 nodes. This happens because, as we increase the number of nodes from 10 to 20 or 40, the file transfer time increases faster than the performance improvement in the algorithm execution.

As previously stated, we intend to study other alternatives to data access. It is important to notice, nonetheless, that the time to distribute the input file will is expected to be diluted as we consider all thesaurus creation steps.

5. Related Work

We have found two other research works that use parallelism to decrease the time of distributional thesauri creation. Riedl and Biemann [8] have applied MapReduce - through Apache Hadoop - and pruning to DT creation. The same authors further explore the subject in another work [2]. The first difference between their work and ours is that they have used Hadoop, while we have used Spark. Spark is believed to be several times faster than Hadoop. Moreover, the main difference is that their works focus on NLP aspects and do not present a performance evaluation. We contribute with a high performance computing point of view by studying performance, speedup, efficiency and discussing the data access strategy.

	Parallel (10 nodes, 20 cores)	Parallel (20 nodes, 40 cores)	Parallel (40 nodes, 80 cores)
Time (s)	1466.34	1499.45	1670.47
Speedup	9.56	9.35	8.39
Efficiency	0.47	0.23	0.10

Table 7. Results considering the transfer file time with the 11GB corpus

6. Conclusions and future work

In this paper we have presented the parallelization with Spark of the word and context counting algorithm, one of the steps of a distributional thesaurus creation. The performance of the parallel version was evaluated in a cluster using up to 40 nodes. The obtained results have shown performance improvements of up to 77%, observed with the largest input file.

This work was conducted in the context of a recently established collaboration between the Parallel and Distributed Processing Group and the Neurocognition and Natural Language Processing Group. Along with other team members, future work focuses on the parallelization and evaluation of other thesauri creation phases: calculation of association between words and contexts and calculation of similarity between words. When the complete pipeline for thesaurus creation is complete, a comparison with another implementation of parallelization [8] will be done. Moreover, as previously discussed, we plan to study the impact of data storage and access strategies in the parallel implementation performance.

Ultimately, a faster implementation for thesauri creation will aid the Neurocognition and Natural Language Processing Group to conduct its research more efficiently. It will allow for more data to be processed, possibly leading to better results quality. Generated code will be publicly available so other researchers in the field can benefit from our results.

7. Acknowledgements

The experiments presented in this paper were carried out on the Grid'5000 experimental test bed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`).

References

[1] M. Baroni and A. Lenci. Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics*, 36(4):673–721, 2010.

[2] C. Biemann and M. Riedl. Text: now in 2d! A framework for lexical expansion with contextual similarity. *J. Language Modelling*, 1(1):55–95, 2013.

[3] A. Ferraresi, E. Zanchetta, M. Baroni, and S. Bernardini. Introducing and evaluating ukwac, a very large web-derived corpus of english. In *Proceedings of the 4th Web as Corpus Workshop (WAC-4) Can we beat Google*, pages 47–54, 2008.

[4] Z. S. Harris. Distributional structure. *Word*, 1954.

[5] R. Lämmel. Googles mapreduce programming modelrevisited. *Science of computer programming*, 70(1):1–30, 2008.

[6] D. Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 2*, ACL '98, pages 768–774. Association for Computational Linguistics, 1998.

[7] M. Padró, M. Idiart, A. Villavicencio, and C. Ramisch. Nothing like good old frequency: Studying context filters for distributional thesauri. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014) - short papers*, Doha, Qatar, Oct. 2014.

[8] M. Riedl and C. Biemann. Scaling to large3 data: An efficient and effective method to compute distributional thesauri. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 884–890, 2013.

[9] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the sun network filesystem. In *Proceedings of the Summer USENIX conference*, pages 119–130, 1985.

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

Managing Healthcare in Hospitals using Ubiquitous Computing Concepts

Cristiano André da Costa, Rodrigo da Rosa Righi, Jorge Luis Victoria Barbosa
Programa de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950 São Leopoldo, Brazil
{cac, rrrigh, jbarbosa}@unisinos.br

Abstract

The inefficiency of the healthcare sector in Brazil can be reinforced by the current use of Information and Communication Technologies (ICT) in health providers. A factor that corroborates with this statement is that only a quarter of those providers employ fully electronic health records. In this way, we propose uHospital, a model to manage healthcare in hospitals. The focus is on the Electronic Health Record (EHR) using the concepts of mobile and ubiquitous computing applied to the health area. These ideas have been defined by the scientific community using two denominations: ubiquitous health, the monitoring of patients health anywhere and anytime, and ubiquitous healthcare, convenient services to patients that allows the clinical diagnosis. To employ these concepts, the model proposes the development of a Personal Health Record (PHR), storing all individual information related to a person's health. This PHR should include exams, diagnoses and also data inserted manually. One possibility in this area, from the popularization of mobile devices and the growth of body sensors use (wearable computing), is to allow people to interact with their PHR, using tablets and smartphones, combining the stored information in their health record with vital signs that have been constant monitored. Particularly, this project proposes the use of situation awareness to combine the patients context, including data being constantly monitored, with information already available in their electronic health record. To fulfill this goal, uHospital employs ontologies based on international and established healthcare standards. In this way, the proposed model allows the management of ubiquitous healthcare in hospitals, centered in a PHR, which permits the inference of patients risk situations.

1. Introduction

The Brazilian healthcare sector is inefficient in a general sense. One of the factors that reinforce this affirmation is the lack of sufficient hospitals to provide good quality service for the population. According to IBGE[1], Brazil had in 2009 2.26 hospital beds per thousand inhabitants, a ratio that has been decreasing in the last 20 years (in 1990 the ratio was 3.71 hospital beds per thousand inhabitants). One obvious alternative would be to build more hospitals or increasing the number of hospital beds in existing establishments. Another way would be to invest in Information and Communication Technology (ICT) in order to improve the efficiency in hospitals.

Just to give an idea of the current situation of ICT in Brazilian healthcare establishments, lets take a look at a survey conducted by the Center for Studying Information and Communication Technology (Cetic.br) in 2013, named TIC Saúde[2]. One factor to demonstrated the inefficiency in terms of ICT in Brazil was that only 25% of health establishments had totally electronic health records for storing patient's information. Furthermore, 9% of all hospital did not had at least Internet access in 2013.

Although this was observed in Brazil, internationally the use of some form of electronic health record for storing patient's information has been used in hospitals in the last years. The industry of Electronic Health Record (EHR) grew 15% in 2012, being estimated in 20.7 billion dollars [16]. However, in the same year, one survey had shown that 39% of physicians did not recommend the EHR they usually use to their colleges [16].

Among the main problems in the current EHRs we highlight the difficulty in integrating and consolidating data among different provides, considering that people uses different clinics, hospitals and laboratories [2]. Another issue is that besides storing patient's data, each EHR creates its

1 http://seriesestatisticas.ibge.gov.br/series.aspx?vcodigo=MS33
2 http://www.cetic.br/saude/2013/

own mechanism of security, requiring a specific set of credentials, such as user and password, locally managed by patients [5]. In a nutshell, the focus of traditional EHR generally is not the patient, but rather the relation between patients and the specific health provider.

In order to contribute minimizing this problem, this article presents the general ideas behind uHospital, a model to manage healthcare in hospitals. The main focus of the project is to propose a Personal Health Record (PHR) allowing patient's to control their medical history and symptoms, storing many information related to their personal health [2]. This PHR should include exams, diagnoses and any other health information possible related.

One possibility in this area, from the popularization of mobile devices and the growth of body sensors use (wearable computing), is to allow people to interact with their PHR, using tablets and smartphones, combining the stored information in their health record with vital signs that have been constant monitored. To allow this combination, uHospital proposes the use of situation awareness [1] to combine the patient's context, including data being constantly monitored, with information already available in their electronic health record. To fulfill this goal, uHospital employs ontologies based on international and established healthcare standards, including OpenEHR [3] and HL7 [4].

The article is further organized in three sections. Section 2 describes some background concepts. A general view of the model is shown in section 3. Finally, section 4 wraps up the article presenting some conclusions and directions for future work.

2. Background

Personal Health Record (PHR) allows individual control of all medical related information [2]. Some studies have shown that allowing people to manage and control their medical data, fosters the patient's cooperation with treatments and makes them more involved in the process [19]. Furthermore, the patient's role in the interaction with their PHR is bond to radically change with the widespread used of mobile devices. Not only they could be used as the means for accessing the information, but rather they can be integrated with sensors, allowing the gathering of health indicators, such as vital signs. The use of wearable and mobile devices allows real-time monitoring of people permitting the correlation of different data and a more proactive action [3]. Consequently, the PHR will be the combination of patient's health record, obtained from interaction with health providers, with data gathered from sensors or wearable devices, using user's smartphone [13] [12].

3 http://www.openehr.org

4 http://www.hl7.org

This scenario directs to the concept of ubiquitous health, consisting of using mobile and ubiquitous computing [6] concepts to monitor patients' health anywhere and anytime, without the need of having them physically presented in clinics and hospitals. The idea of maintaining pervasive medical care, has been called ubiquitous healthcare, consisting in providing a convenient service to patients, easing the diagnostic of clinical conditions, improving efficiency, accuracy, and availability of medical treatment [10].

The convergence of ubiquitous healthcare with PHR come into sight from three factors: the natural evolution of information access from mobile devices, the inherent mobility of those devices, and the integration of assorted sensors [12]. This convergence may bring ethical challenges, such as data privacy, and several research opportunities, given the physical limitations of mobile devices (in terms of memory, energy and power processing), compatibility between different platforms, and handling the large amount of data that could be gathered from sensors [3] [12].

3. uHospital Model

The uHospital model explore the concepts of ubiquitous health and ubiquitous healthcare in hospitals. The proposal consists in defining a model for managing health in hospitals using many concepts related to mobile and ubiquitous computing. Among the employed concepts, we highlight the use of information related to the users, such as their vital signs. This idea, named context awareness, consists in using available information regarding people and their surrounding [7] [8]. Particularly, the proposal focus on the use of a special kind of context awareness named situation awareness, in which many types of context are aggregated to generate a more complex view of the circumstance [18]. The idea of using situation awareness is to interact with users, learn their behavior, using information obtained from physical and virtual sensors, and making autonomic actions according to the circumstance detected [1].

The model central focus is on electronic health record, which is identified as one of the main challenges to be addressed by the emergent ubiquitous technologies to the health area [15] [17]. More specifically, the model proposes a PHR accessible from mobile devices, using the situation awareness in order to detect possible risks to patient's health. Using this information, health providers can act in a proactive way. Some additional characteristics have been used in the proposal to address some other current limitations in PHR proposals. To foster interoperability with others PHRs and EHRs, the solution is based on a set of international standards, including the already mentioned OpenEHR and HL7. Particularly, we are employing the HL7 Clinical Docu-

ment Architecture (CDA)[5], the set of characteristics defined by the Joint Electronic Personal Health Record Task Force [11], the ISO/TR 20514:2005[6], which defines informations that should be present in EHRs and also the more recent ISO/TR 14292:2012[7], which specifically covers Personal Health Records. In the model, PHR informations are stored in an ontology allowing the use of inferences to detect possible situations and defining involved risks [14] [19].

In terms of architecture, the model uses the concept of mobile cloud computing, in which part of the data and processing is done out of the mobile device, allowing to surpass the physical limitations of those equipments [9]. To deal with privacy and security, further improving the interoperability with different providers, the model uses OpenID, an international standard to create identities and distributing credentials employed by many Internet providers [5] [4].

In this way, the present model aims at answering the following research question: *How is it possible to improve the efficiency of health management in hospitals, by storing informations in a Personal Health Record, using the concepts of ubiquitous health and ubiquitous healthcare?*

4. Conclusion

This article presented uHospital, a model allowing the management of ubiquitous healthcare in hospitals, centered in a PHR, which permits the inference of patients risk. In the context of uHospital, the group of researchers at Unisinos is currently advising many dissertations and thesis. Although recent, the project fosters the relation with Sistema de Saúde Mãe de Deus, a group of hospitals located in the Rio Grande do Sul state and also plans to contribute to minimizing the efficiency of Brazilian Health area.

As a future work, we are in the process of developing a prototype of a PHR service based on cloud computing, and also clients for acessing it from iOS and Android smartphones. Furthermore, we are currently experimenting with the use of mobile and wearable sensors and investigating ways of detecting risks to patient's health regarding different pathologies, including heart fail, mental health and chronicler diseases.

5. Acknowledgment

The authors would like to thank to CNPq, CAPES and FAPERGS for financing this research. We also thank to Sistema de Saúde Mãe de Deus and Value Health company for collaborating in this work.

5 http://goo.gl/KjSqkq
6 http://goo.gl/F069V0
7 http://goo.gl/XZk5K5

References

[1] C. B. Anagnostopoulos, Y. Ntarladimas, and S. Hadjiefthymiades. Situational computing: An innovative architecture with imprecise reasoning. *Journal of Systems and Software*, 80(12):1993–2014, 2007.

[2] K. Belyaev, I. Ray, and G. Luckasen. Personal health record storage on privacy preserving green clouds. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*, pages 448–457. IEEE, 2013.

[3] Y. Chen, K. Cheng, C. Tang, K. A. Siek, and J. E. Bardram. Is my doctor listening to me?: impact of health it systems on patient-provider interaction. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 2419–2426. ACM, 2013.

[4] B. Coats and S. Acharya. The forecast for electronic health record access: partly cloudy. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 937–942. ACM, 2013.

[5] B. Coats and S. Acharya. Leveraging the cloud for electronic health record access. *Perspectives in Health Information Management*, 11(Winter), 2014.

[6] C. A. da Costa, A. C. Yamin, and C. F. R. Geyer. Toward a general software infrastructure for ubiquitous computing. *IEEE Pervasive Computing*, (1):64–73, 2008.

[7] A. K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[8] W. Du and L. Wang. Context-aware application programming for mobile devices. In *Proceedings of the 2008 C 3 S 2 E conference*, pages 215–227. ACM, 2008.

[9] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[10] Y. E. Gelogo and H.-K. Kim. Unified ubiquitous healthcare system architecture with collaborative model. *International Journal of Multimedia and Ubiquitous Engineering*, 8(3):239–244, 2013.

[11] D. A. Jones, J. P. Shipman, D. A. Plaut, and C. R. Selden. Characteristics of personal health records: findings of the medical library association/national library of medicine joint electronic personal health record task force. *Journal of the Medical Library Association: JMLA*, 98(3):243, 2010.

[12] H. Kharrazi, R. Chisholm, D. VanNasdale, and B. Thompson. Mobile personal health records: an evaluation of features and functionality. *International journal of medical informatics*, 81(9):579–593, 2012.

[13] S. KSimon, K. Sonai Muthu Anbananthen, and S. Lee. A ubiquitous personal health record (uphr) framework. In *2013 International Conference on Advanced Computer Science and Electronics Information (ICACSEI 2013)*. Atlantis Press, 2013.

[14] M. Peleg, T. Broens, A. González-Ferrer, and E. Shalom. Architecture for a ubiquitous context-aware clinical guidance system for patients and care providers. *KR4HC'13/ProHealth'13*, pages 161–167, 2013.

[15] M. Rigby. Applying emergent ubiquitous technologies in health: The need to respond to new challenges of opportunity, expectation, and responsibility. *International journal of medical informatics*, 76:S349–S352, 2007.

[16] A. Schutzbank and R. Fernandopulle. Doubling down: Lessons learned from building a new electronic health record as part of primary care practice redesign. In *Healthcare*, volume 2, pages 14–18. Elsevier, 2014.

[17] F. Senne, A. Barbosa, W. Oyadomari, and A. Bittencourt. For challenges of e-health policies in brazil: An analysis availability and use of icts at premises brazilian health. In *CPR LATAM-Communication Policy Research Conference*, 2014.

[18] S. Stipkovic, R. Bruns, and J. Dunkel. Pervasive computing by mobile complex event processing. In *e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on*, pages 318–323. IEEE, 2013.

[19] A. K. Triantafyllidis, V. G. Koutkias, I. Chouvarda, and N. Maglaveras. A pervasive health system integrating patient monitoring, status logging, and social sharing. *Biomedical and Health Informatics, IEEE Journal of*, 17(1):30–37, 2013.

Avaliação de um Provedor de Nuvem de Serviços para Aplicações Distribuídas Intensivas em Dados e em Computação

Raffael B. Schemmer, Cláudio F. R. Geyer
Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, RS, Brasil
{raffael.schemmer, geyer}@inf.ufrgs.br

Resumo *- Este trabalho apresenta um estudo comparativo de dois tipos de serviços de cloud computing, para execução e processamento de aplicações intensivas em dados e em computação. Utiliza-se um conjunto de máquinas virtuais no nível de infraestrutura como serviço (IaaS) e de plataforma como serviço (PaaS). O trabalho faz uso do provedor de serviço de cloud computing Microsoft Azure. Os resultados apontam que o serviço de IaaS apresenta o melhor desempenho frente ao tempo de execução. O serviço de PaaS é aquele que apresenta a melhor aceleração, quando mais de uma máquina virtual é utilizada. No que trata dos custos, a infraestrutura como serviço (IaaS) apresentou-se como melhor solução.*

1. Introdução

Pesquisas no campo da computação e do uso de tecnologias e equipamentos digitais, apontam que no intervalo de tempo entre os anos de 2005 a 2020, o universo digital irá crescer em uma ordem de 300 vezes [1]. O investimento em infraestrutura de TI irá crescer na mesma ordem de grandeza e proporção. O custo pelo armazenamento e processamento de um Gigabyte de informação irá decair na ordem de 100 vezes [1]. Estas mesmas pesquisas apontam que em 2020, 40% de toda informação processada e manipulada será feita por máquinas do tipo cloud computing [1]. Estes indicadores reforçam a pesquisa por tecnologias capazes de realizar a captura e o processamento de grandes quantidades de dados. Ainda, tecnologias que consigam operar em máquinas do tipo cloud computing, estando aptas para lidar com questões específicas de desempenho existentes neste tipo de infraestrutura.

Este trabalho foi motivado inicialmente pelo interesse do GPPD (Grupo de Processamento Paralelo e Distribuído) na demanda pelo uso de soluções intensivas em dados utilizando cloud computing. Nas seções a seguir, diferentes soluções de serviços de cloud computing com ênfase no processamento de dados serão detalhadas e estudadas em profundidade. O objetivo deste trabalho é realizar um estudo avaliativo no sentido de quantificar o desempenho e a escalabilidade de dois serviços de cloud computing, com ênfase em aplicações intensivas em dados e em computação. Deste objetivo, espera-se demostrar qual das soluções de serviço apresenta: (*i*) A melhor escalabilidade; (*ii*) O melhor desempenho; (*iii*) A melhor solução entre o desempenho e o custo da infraestrutura.

O trabalho estrutura-se da seguinte forma. Na seção II serão apresentados conceitos e características das tecnologias utilizadas pelo trabalho. Na seção III, explica-se a estratégia adotada para realização da avaliação proposta. Na seção IV são apresentados e discutidos os resultados. Por fim, a seção V apresenta as principais conclusões e trabalhos futuros.

2. Plataformas e Conceitos Básicos

A. O Framework Hadoop

O Hadoop é um ambiente que permite o armazenamento e o processamento de dados de forma distribuída. Dentre os principais componentes que caracterizam o ambiente estão o sistema de arquivos distribuído (HDFS) e o modelo de programação MapReduce (MR). Assume-se neste trabalho que um ambiente de computação distribuída será baseado em um conjunto de instâncias de computadores ou máquinas virtuais, providas por um ambiente de cloud computing. O HDFS é formado por uma arquitetura de serviços do tipo mestre/escravo. O mestre é implementado pelo componente *Namenode*, e os escravos pelo componente *DataNode* [2]. O Hadoop através do HDFS, permite que um conjunto arbitrário de computadores armazenem arquivos de forma distribuída.

O modelo de programação MapReduce (MR), possibilita a implementação de operações de transformação em dados (filtros, agregações, iterações) através das operações elementares do tipo Map e Reduce. Ele assume que toda comunicação de entrada e saída seja feita através da estrutura de elementos chave/valor [2]. A Figura 1 ilustra o funcionamento da técnica.

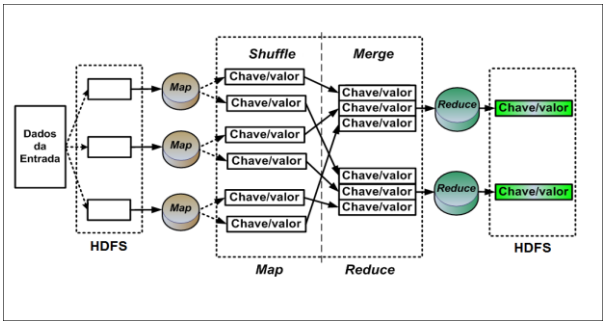


Figura 1: Modelo de programação MapReduce (MR) [6].

Conforme ilustra a Figura 1, a leitura dos blocos dos arquivos do HDFS é feita via processos Map. A escrita no HDFS é feita via processos Reduce. Toda leitura e escrita é realizada através de tuplas de elementos, definidos pela estrutura chave/valor. Os processos Map e Reduce se comunicam ou trocam mensagens através da etapa chamada de *shuffle/merge* [2]. Para cada chave/valor escrita por um Map, um número identificador único é gerado, chamado *hashcode*. Este número é vinculado a um processo Reduce, que irá receber toda chave/valor equivalente a este identificador.

A arquitetura do Hadoop que realiza a coordenação e execução dos processos MapReduce é formada por uma organização de componentes do tipo mestre/escravo. O mestre é implementado pelo componente *JobTracker*, os escravos pelo componente *TaskTracker* [2]. A Figura 2 ilustra a arquitetura dos componentes que formam o Hadoop.

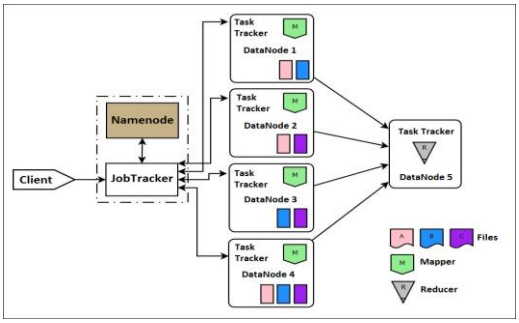


Figura 2: Arquitetura de componentes do Hadoop que compõem o modelo MR e o HDFS [6].

B. A Cloud Microsoft Azure

O conceito de cloud computing tem relação a um modelo de negócios onde um conjunto de recursos de computação são alocados e tarifados, conforme a demanda do usuário [3]. Diferentes são os modelos de serviço de cloud computing existentes na atualidade. Este trabalho fará uso de um modelo de cloud pública, sendo ela a Microsoft Azure, provida e administrada pela Microsoft. Os recursos de computação são providos sob diferentes tipos de modelos de serviço. Este trabalho fará uso do modelo de infraestrutura como serviço (IaaS) e do modelo de plataforma como serviço (PaaS) [3].

No nível de infraestrutura (IaaS), o provedor Microsoft Azure permite que o usuário defina algumas características das máquinas virtuais. O sistema operacional é uma destas características. O usuário poderá escolher entre versões de sistemas Windows ou Linux. A afinidade é outra característica disponível em IaaS. O usuário poderá definir em qual continente suas máquinas virtuais deverão ser executadas [4]. A Microsoft Azure opera hoje em 4 continentes somando ao todo, 19 data centers em operação [4]. A afinidade permite minimizar os atrasos de rede, maximizando o desempenho. O usuário poderá durante a configuração de uma máquina virtual, definir um conjunto de serviços previamente disponíveis em um banco de serviços. Estes serviços serão instalados juntamente com o sistema operacional da máquina virtual.

Em nível de infraestrutura (IaaS), o usuário poderá definir a quantidade de processadores e de memória que cada máquina virtual deverá possuir. A Azure oferece 30 tipos de configurações, classificadas em 3 classes [5]: (*i*) Classe A: Específicas para computação de propósito geral; (*ii*) Classe D: Especificas para computação de alto desempenho; (*iii*) Classe G: Especificas para computação intensiva em dados. Cabe observar que cada máquina virtual irá possuir um conjunto de núcleos ou *cores* de processador, uma quantidade finita de memória RAM, e um disco temporário, utilizado como *cache* de sistema. Este disco não possui réplicas externas. Em caso de falha ou desligamento da máquina virtual falhar, os dados presentes neste *cache* serão perdidos.

Em nível de plataforma (PaaS), o provedor de cloud computing Microsoft Azure oferece alguns tipos de aplicações especificas, sendo uma delas o Hadoop, chamado de HDInsights. O Hadoop é desenvolvido pela Apache como um projeto *Open Source*. O HDInsights é uma versão modificada e comercial do Hadoop, desenvolvida pela empresa HortonWorks para a Microsoft. O HDInsights é oferecido em versões Windows e Linux. Em nível de plataforma (PaaS), o usuário pode apenas definir o número de máquinas virtuais que irão formar o sistema distribuído, e uma conta de armazenamento do tipo Azure blobs [4] utilizada pelo HDInsights para execução do HDFS. Conforme maior é o tipo e o custo da máquina virtual, maior será o desempenho da rede e dos discos Azure blobs [5].

3. Avaliação Proposta

No nível de serviço como infraestrutura (IaaS), este trabalho propôs fazer uso de 4 tamanhos de *cluster* formados por uma, duas, três e quatro instâncias de máquinas virtuais. Criou-se grupos de afinidade, na tentativa de aproximar as máquinas virtuais de cada *cluster* e reduzir a latência de rede.

Foram utilizadas máquinas virtuais do tipo D3, formadas cada uma por 4 processadores Xeon E5-2660 Sandy Bridge-EP, 14Gbytes de memória DDR3 e discos locais SSD de 200Gbytes cada [5]. Para execução dos testes em IaaS foi utilizado o disco *cache* para o armazenamento dos dados. Cada instância de máquina virtual D3 custava no momento da execução dos resultados 0.376 dólares, por hora de utilização [5]. Optou-se quanto ao software, fazer uso do Linux Ubuntu 15.04 64bits, do Java 7 e do Hadoop 1.2.1. A instalação foi realizada de forma manual, conforme documentação base do Hadoop disponível pela Apache e documentada em [2].

No nível de serviço como plataforma (PaaS), optou-se pelo uso da tecnologia HDInsights para Linux, que operava no estado experimental [4]. O serviço exige que uma máquina virtual seja utilizada exclusivamente para executar os serviços de gerência do Hadoop. Este trabalho considera uma máquina virtual adicional para todos os tamanhos de *cluster* utilizados em (PaaS), apenas para execução dos serviços de gerencia do Hadoop. O HDInsights Linux não permite que sejam definidas regiões de afinidade específicas, como utilizadas no nível de infraestrutura (IaaS). Este trabalho definiu apenas que as máquinas virtuais utilizadas fossem mapeadas para executar no data center número 2 da costa leste dos Estados Unidos [5].

Com relação ao tipo de máquina virtual utilizada, este trabalho fez uso das mesmas configurações utilizadas em nível de serviço de infraestrutura (IaaS). O HDInsights não utiliza o disco local *cache* SSD das máquinas do tipo D3, exigindo que o usuário informe uma conta de armazenamento externa do tipo Azure blob [4] para instalação do HDFS e para persistência dos dados do Hadoop. O Hadoop HDInsights Linux disponível e utilizado foi o HDP 2.2 [4], implementação baseada no Hadoop Yarn 2.6. O HDInsights Linux utiliza uma versão Linux Ubuntu 12.04 LTS de 64bits e Java 7. Cada instância de máquina D3 rodando HDInsights custava no momento da execução 0.684 dólares por hora utilizada [5].

Este trabalho fez uso de duas aplicações MapReduce para avaliar os ambientes propostos sendo elas: (i) Aplicação

MapReduce que realiza o cálculo de aproximação do número Pi. Esta aplicação é considerada intensiva em computação, e não realiza leituras no HDFS. Este trabalho propôs utilizar um milhão de casas após a vírgula para o cálculo do Pi, e 50 Maps geradores de números; (ii) Aplicação MapReduce que realiza a contagem de palavras é considerada intensiva em dados e em computação, pois realiza leituras e escritas intensivas no HDFS. Este trabalho propôs utilizar um arquivo de 1.28Gbytes de tamanho, gerado a partir da função *random* do Linux para uma faixa de valores inteiros de 1 milhão de valores. Ambas as aplicações utilizadas fazem parte da biblioteca de códigos exemplo disponíveis em ambas as versões do Hadoop. A quantidade de aplicações, tamanho das entradas dos problemas e a quantidade de recursos foram escolhidos em função do custo e das restrições impostas pelo provedor de serviços.

O cálculo do Pi irá verificar a capacidade intensiva na realização de cálculos das plataformas utilizadas. A contagem de palavras fará uso intensivo do sistema de entrada e saída das máquinas, formados pelos discos rígidos e pela rede. Para cada um dos resultados, foi aplicada uma normalização da média do somatório de 10 execuções. Este trabalho não considera o tempo para escrita dos dados no HDFS nem o tempo de criação das máquinas virtuais. Como métricas serão utilizadas: (*i*) O tempo de execução em segundos, com objetivo de demostrar qual das plataformas possui o melhor desempenho; (*ii*) A aceleração, utilizada para demostrar qual das plataformas terá a melhor escalabilidade; (*iii*) A multiplicação do custo das máquinas pelo tempo de execução, em uma tentativa de demostrar a diferença existente entre o custo e o desempenho.

4. Resultados

A primeira métrica avaliada será o tempo de execução para a aplicação de contagem de palavras. O conjunto de máquinas virtuais foi variado entre uma, duas, três e quatro máquinas tanto para o serviço de infraestrutura (IaaS) como de plataforma (PaaS). Observa-se na Figura 3 os seguintes resultados: (*i*) O serviço (IaaS) apresenta um tempo de execução inferior ao do serviço (PaaS), que varia em um percentual de 30% a 40% dependendo o número de máquinas virtuais utilizadas; (*ii*) Ambos os serviços apresentam redução no tempo de execução conforme mais de uma máquina virtual é utilizada. Nota-se uma redução expressiva no tempo de execução quando apenas duas máquinas virtuais são utilizadas. Há porém um pequeno ganho no tempo de execução, quando mais de duas máquinas virtuais são utilizadas.

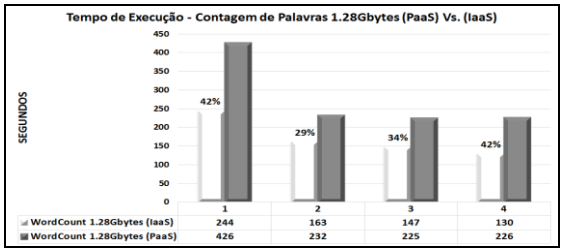


Figura 3: Tempo de execução para a aplicação MapReduce de contagem de palavras.

Ambos os serviços de plataforma (PaaS) e infraestrutura (IaaS) foram variados de maneira similar a contagem de palavras para a aplicação do cálculo do Pi. Observa-se a partir dos dados apresentados na Figura 4 que ambas as plataformas reduziram o tempo de execução da aplicação, que acontece de maneira mais expressiva quando uma segunda máquina virtual é utilizada. Observa-se também uma diferença no tempo de execução entre os serviços de infraestrutura (IaaS) e de plataforma (PaaS), da ordem de 70% a 80%.

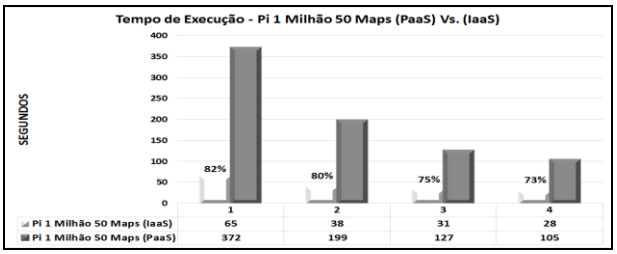


Figura 4: Tempo de execução para a aplicação MapReduce do cálculo do Pi.

Este trabalho calculou o desvio padrão para cada tamanho de *cluster* em cada um dos diferentes tipos de serviços avaliados. Para os quatro tamanhos de *cluster*, foi calculada a média do somatório de valores. Para a aplicação de contagem de palavras, o somatório do desvio padrão dos quatro tamanhos de *cluster* apontou que o serviço de (PaaS) apresentou 36% menor variação entre os resultados aferidos. Já para a aplicação referente ao cálculo do Pi, o serviço de (IaaS) apresentou 61% menor variação entre os resultados obtidos. Estes dados indicam que mesmo o (PaaS) apresentando resultados superiores quanto a piora no tempo de execução, o provedor de serviços dedica esforços em oferecer um serviço capaz de oferecer a menor variabilidade possível para aplicações intensivas no uso do disco e da rede. Já aplicações intensivas em computação, como o cálculo do Pi, o cenário (IaaS) por não depender de entrada e saída demostrou melhor desempenho.

A segunda métrica avaliada será a aceleração. Esta métrica correlaciona o tempo sequencial, sendo ele o tempo de execução em uma máquina virtual, pelo tempo paralelo quando mais de uma máquina virtual é utilizada. A Figura 5 apresenta a aceleração obtida para a aplicação de contagem de palavras. O número de máquinas virtuais é variado em uma, duas, três e quatro máquinas. Observa-se pelo gráfico que o serviço (PaaS) obtêm a melhor aceleração, independente que o tempo de execução seja superior ao serviço de (IaaS).

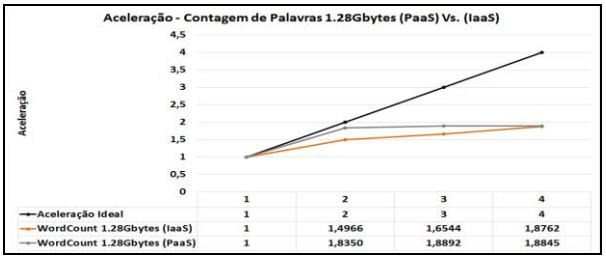


Figura 5: Fator de aceleração para a aplicação MapReduce de contagem de palavras.

A aceleração também é avaliada para a aplicação do cálculo do Pi. A Figura 6 apresenta os resultados obtidos. Observa-se que a plataforma (PaaS) obtêm a melhor aceleração, ficando próxima do ideal em grande parte dos casos avaliados. Semelhante a aplicação de contagem de palavras, o tempo de execução do serviço de (PaaS) é superior ou pior, comparado ao (IaaS). A análise do desvio padrão, aponta que o provedor de serviço no nível de (PaaS), procura oferecer uma plataforma

mais robusta, mas deixa a desejar quanto a melhor desempenho, sendo ele o tempo de execução.

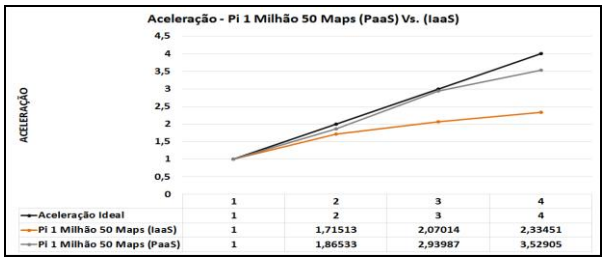


Figura 6: Fator de aceleração para a aplicação MapReduce do cálculo do Pi.

As respectivas Figura 7 e Figura 8, apresentam a multiplicação do tempo de execução de cada aplicação, pelo custo da máquina virtual de cada tipo de serviço avaliado. Os resultados apontam que para a aplicação de contagem de palavras, o custo do serviço de (PaaS) oscila entre 70% a 90%, conforme a quantidade de máquinas virtuais utilizadas. A diferença acontece, pois o custo da instância de máquina virtual no serviço de (PaaS) é 181% maior que o serviço (IaaS). Mesmo que a diferença no desempenho exista, o custo de 181% para o serviço de (PaaS) torna esta solução não atrativa em relação ao desempenho pelo custo associado.

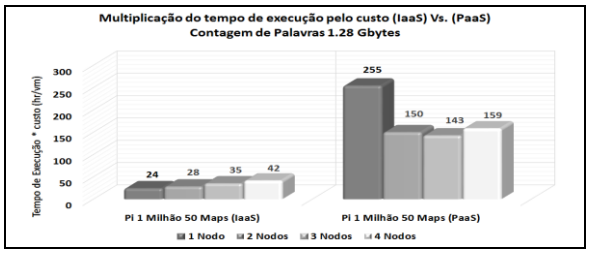


Figura 7: Multiplicação do tempo de execução da aplicação de contagem de palavras pelo custo.

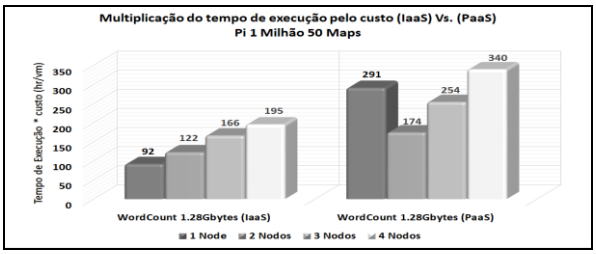


Figura 8: Multiplicação do tempo de execução da aplicação do cálculo do Pi pelo custo da máquina virtual.

Em relação a aplicação do cálculo do Pi, a diferença entre o custo do serviço de (PaaS) oscila entre 30% a 70%, dependendo a quantidade de máquinas virtuais utilizadas. O serviço de (PaaS) apresentou excelente aceleração, demostrando que para aplicações Hadoop intensivas em computação, e que necessitem de um número expressivo de máquinas virtuais, o serviço de (PaaS) é aquele capaz de obter escalabilidade necessária, sendo este a melhor solução aferida.

Os resultados de todas as métricas avaliadas demostram que no nível de serviço (PaaS), o provedor de serviços fará um esforço para prover recursos físicos com alto grau de precisão e baixa variabilidade. Esta qualidade no serviço virá acrescida de um custo adicional no serviço. O desvio padrão e a aceleração das execuções demostram este comportamento. O tempo de execução por outro lado acaba sendo maior, em função do tipo de ambiente Hadoop utilizado pelo HDInsights e pelas técnicas de administração adotadas pelo (PaaS). Este conjunto de informações divergentes abrem campo para inúmeros tipos de pesquisa e de técnicas a serem implementadas e estudadas.

5. Conclusões e Trabalhos Futuros

Este trabalho apresentou, um estudo de dois tipos de serviços de cloud computing, sendo um deles no nível de infraestrutura (IaaS) e outro no nível de plataforma (PaaS), onde duas aplicações intensivas em computação e em dados foram avaliadas. Apresenta-se uma série de resultados que demostram as capacidade de cada um dos ambientes avaliados. No que trata do tempo, o serviço de IaaS foi aquele que apresentou o melhor desempenho em ambas as aplicações, sendo este o mais bem preparado para uso em pesquisas pelo GPPD no futuro. No que trata da métrica de aceleração, o serviço de PaaS apresenta-se como melhor opção, devendo ser considerado em situações onde várias dezenas e/ou centenas de máquinas virtuais sejam necessárias. Por fim, o trabalho avalia a questão do tempo de execução pelo custo da máquina virtual. Observa-se em ambos os casos, que o serviço de IaaS oferece para as aplicações avaliadas o menor custo e o melhor desempenho, em razão do custo adicional do HDInsights.

Uma série de trabalhos futuros surgem a partir deste estudo realizado. O trabalho utiliza o Hadoop 1.2.1, recomenda-se que trabalhos futuros avaliem o efeito do uso do Hadoop 2 Yarn e suas diferenças no desempenho. Trabalhos futuros devem considerar o uso de serviços Azure blobs [5] [4] no serviço IaaS, e não o uso do *cache* local SSD. O desvio padrão deve ser avaliado nesta situação. A Azure oferece outros tipos de máquinas, recomenda-se que o tipo G [5], seja utilizado. A aceleração demostrada pelo serviço de PaaS deve ser avaliada para um número maior de máquinas virtuais.

Agradecimentos

Os autores deste trabalho agradecem ao GPPD (Grupo de Processamento Paralelo e Distribuído) por fornecer espaço e subsídio para o desenvolvimento deste trabalho de pesquisa. Também, a Microsoft Research por fornecer acesso e incentivos para uso da cloud computing Microsoft Azure através do projeto Microsoft Azure for Research.

Referências

[1] Gantz, J; Reinsel, David. “THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadow s, and Biggest Grow th in the Far East”. Technical Report 2012.

[2] White, T. (2012). Hadoop - The Definitive Guide, volume 1. OReilly Media, Inc., 3rd.

[3] Peter M. Mell and Timothy Grance. 2011. SP 800-145. The NIST Definition of Cloud Computing. Technical Report. NIST, Gaithersburg, MD, United States.

[4] Chauhan, A; Fontama, Woody, B. “Introducing Microsoft Azure HDInsight”. Microsoft 2015.

[5] Microsoft. “How Azure pricing works”. Disponível em: http://azure.microsoft.com/en-us/pricing/details/virtual-machines/. Acessado em : Agosto de 2015.

[6] Schemmer, R.B., Anjos, J.C.S., Tibola, A.L., Barros, J.F., Geyer, C.F.R.; Framework Hadoop em Plataformas de Cloud e Cluster Computing. Escola Regional de Alto Desempenho (ERAD). pp.71,88. Abril 2015.

Performance Analysis of a Numerical Weather Prediction Application in Microsoft Azure

Emmanuell D. Carreño, Eduardo Roloff, Jimmy V. Sanchez, and Philippe O. A. Navaux

Federal University of Rio Grande do Sul, Informatics Institutemendmend
Porto Alegre, Brazil
{edcarreno, eroloff, jkmvsanchez, navaux}@inf.ufrgs.br

Abstract

The Cloud Computing paradigm emerged as a practical solution to perform large-scale scientific computing. The elasticity of the Cloud and its pay-as-you-go model presents an attractive opportunity for applications commonly executed in clusters or supercomputers. In this context, the user does not need to make an upfront purchase of infrastructure. The resources can be rented from a provider to be used for a period of time. In this paper, we present the performance results of executing and scaling horizontally a numerical weather prediction application in a Cloud infrastructure. Results show that scalability and performance are acceptable and that the Cloud Computing paradigm represents an attractive environment for high performance applications.

1. Introduction

Computer generated weather forecast is a continuous developing area that has been executed mostly on grids or supercomputers. This is because applications in climate and weather research typically require huge amounts of data and computing power [1]. Numerical Weather Prediction (NWP) has been benefited by the computing area for years, the use of computers has allowed them to use more variables in their simulations and more data sources to perform precise predictions. This computing approach has achieved much, but can also be a significant limitation to progress [2].

The costs associated with hardware investment and the need for energy efficiency have motivated research in the usage of new technologies to optimize costs; Cloud Computing is one of them. Cloud Computing provides cost-effective, fast, and a vast amount of resources for large-scale applications. The Cloud is used as utility where computing, storage, and networking services are provided on-demand.

On-demand Cloud Computing offers an attractive new dimension to High Performance Computing (HPC), in which virtualized resources can be appropriated. These resources could be used in a customized manner, to target a particular scenario, at the time and in the form the user desires.

Some HPC initiatives have started utilizing Cloud Computing for research. However, it is clear that one of the major concerns for the HPC community is the performance [3]. For weather research, the need to reduce the waiting time found in the batch jobs schedulers common in HPC infrastructure, the possibility of quickly access and share the processed data, and the development of newer simulations or ensemble forecasts has motivated them to use the Cloud.

2. Numerical Weather Prediction

Numerical Weather Prediction (NWP), is the use of mathematical models to predict, accurately, the atmospheric behavior for an specific period. It represents the state of the atmosphere at a fixed time (temperature, wind components, etc.) over a discrete domain [1]. The results are denominated weather forecast when predicts only a few days and climate forecast when predicts months.

BRAMS was the NWP evaluated in this work. BRAMS is one of the most widely employed numerical models in regional weather centers in Brazil [4]. It is a model based on RAMS [5]. RAMS, developed by the atmospheric science department at the Colorado State University, is a numerical weather prediction model used to simulate atmospheric conditions. The new developments integrated into BRAMS include modified computational modules to better simulate the tropical atmosphere. The primary objective of BRAMS is to provide a single model for Brazilian regional weather centers.

3. Microsoft Azure

Microsoft started its initiative in Cloud Computing with the release of Windows Azure at its annual professional developer conference in 2008. The initial model was the platform as a service allowing developing and running applications written in the programming languages supported by the .NET framework. Nowadays, the Azure infrastructure covers all types of service models and has gained a significant market share. The service prices vary from location to location. Because of this factor, selecting the best price for each application can reduce or increase costs significantly.

We used the Infrastructure as a Service (IaaS) model, one of the Cloud service models. This model provide us complete management capabilities over the Virtual Machines (VMs) running on Microsoft's infrastructure. Azure offers multiple options regarding IaaS [6]. Each Virtual Machine configuration is called *instance size* using Azure's conventions. Some of the instances sizes are very interesting for HPC, especially the ones with Infiniband. Unfortunately, Infiniband was available only for Windows virtual machines using a proprietary MPI driver. We selected only virtual machines using Ubuntu Linux distribution, therefore we did not used any Infiniband for our experiments.

Microsoft Azure was chosen because previous work [7] has shown that performance and cost efficiency for HPC in Azure was better than on Rackspace and Amazon.

4. Methodology

The tests were performed by scaling the number of nodes available to the application. The experiments consisted of executing BRAMS to perform a forecast of 72 hours with a spatial resolutions of 20 km and 5 km covering a an area of 1500×1500 km. Each forecast simulation was performed five times using virtual machine instances with eight CPU cores. Table 1 contains the specifications of the VM instances used on the experiments. At every iteration, we added a new instance up to eight nodes (168 CPU cores).

Characteristic	Cloud Instance Size			
	A4	A7	A8	A9
Processor Model	Xeon E5-2660		Xeon E5-2670	
Processor Speed (GHz)	2.2		2.6	
Number of CPU Cores	8	8	8	16
Memory (GB)	14	56	56	112
Networking (Mbps)	800	2000	5000	10000

Table 1. Specifications of the Cloud instances used for the performance experiments.

All the experiments were executed in the same Azure datacenter location using Affinity Groups. Using this feature we could achieve better networking performance as the VM would be as physically close to each other as possible. The datacenter we selected was Europe North.

Scalability is used to establish the possibility of an application to use more resources in case it needs to use them. This could be seen on two different axes, horizontal or vertical scaling. In the Cloud, horizontal scaling is referred to the possibility of adding more virtual machines for processing. The adding process could reach a limit in which adding more machines causes only the increasing in experiment costs without increasing the performance.

Vertical scaling in the Cloud is used commonly to upgrade the characteristics of the instance running a service. Usually, this process occurs on-demand and over a running instance. BRAMS uses homogeneity to balance resource us-

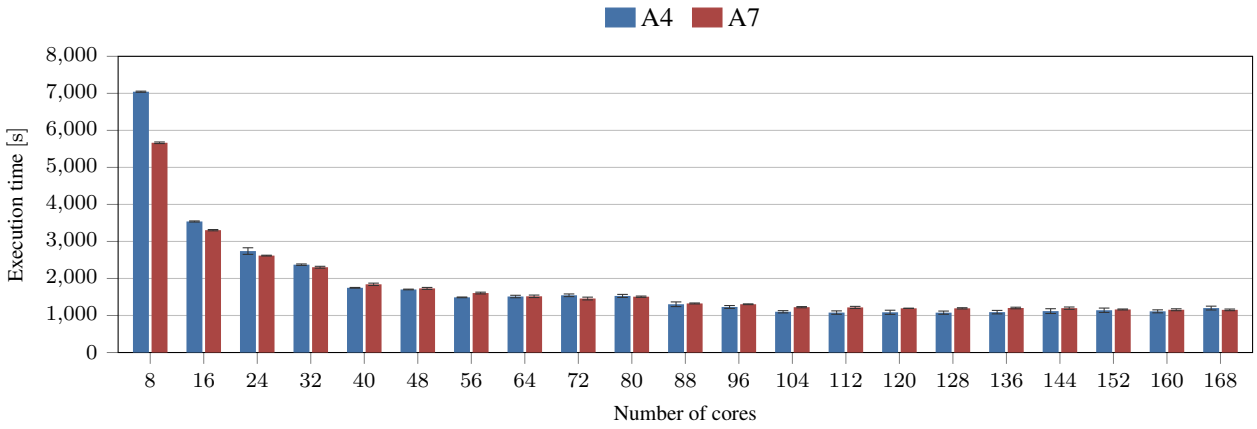


Figure 1. Scaling experiments with 168 cores using a 20 km model resolution with A4 and A7 VM instance sizes. Error bar indicates standard deviation.

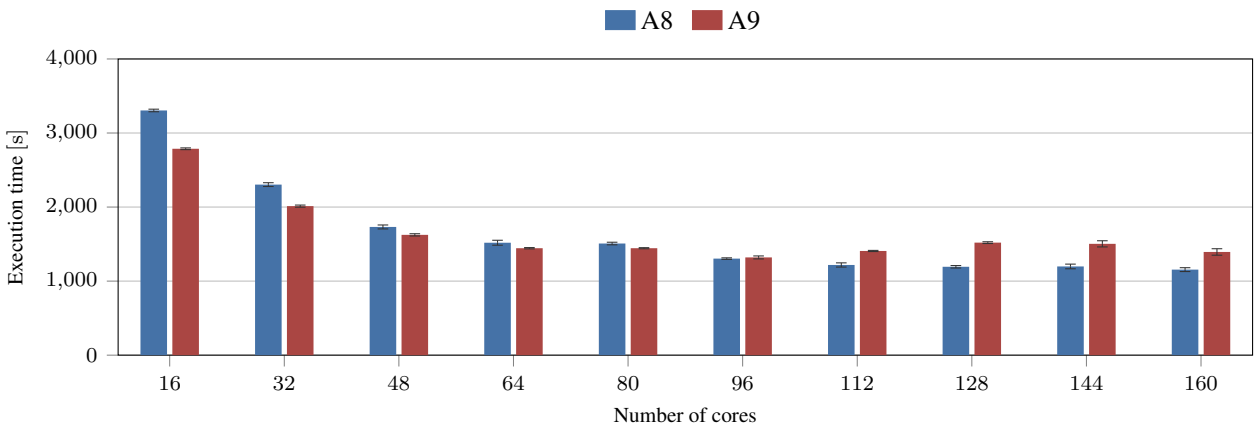


Figure 2. Experiments with 160 cores and a 5 km model resolution using A8 and A9 VM instance sizes. Error bar indicates standard deviation.

age. Due to this behavior is not wise to upgrade the instance characteristics in the middle of an experiment while running the application. Our vertical scaling experiments were based on using Cloud instances with different characteristics, but maintaining the same configuration for all the machines during each experiment. Using this approach we could measure in which point was not possible to get more performance gains scaling BRAMS in the Cloud.

5. Results

We performed the first scaling experiments using standard A4 and A7 instance sizes. As depicted in Figure 1 the application was not able to scale well beyond 88 cores. Even using that amount of cores is not worth, over 56 cores the time reduction is less than two percent at each scaling step.

Generally speaking the performance gets worst as more nodes are added to the experiment for both instance sizes. This can be attributed to the fact that BRAMS is an application with a high amount of communication between cores and nodes. Other factors influencing the scalability could be related to network latencies or access to the Cloud storage implemented while migrating [8] BRAMS to Azure.

We evaluated if the reason why BRAMS did not scale was because the instances used were not the ones with the best performance. We performed another pair of experiments using the instance sizes with the best networking capabilities.

The results are shown on Figure 2. This experiment was performed using A8 and A9 instances; the A8 instance supports eight virtual cores with 56GB of memory while the A9 instance supports 16 virtual cores with 112GB of memory. Due to the difference in the number of cores, we run the forecasts only using 16 cores at a time for the A9 instance or pairs of A8 instances.

In this case, BRAMS behaves a little different but also confirms that scalability was limited. The application scaled correctly to near 80 cores. After that point, performance worsened with each node added using A9 instances. In the case of A8 instances, after 88 cores performance gains were low.

It is interesting to notice that an A8 instance is similar to an A9 in most of the characteristics but the number of cores. However, A8's performance results and scaling were better than A9's. Besides, A9's price is twice the price of an A8. The results show that in the case of BRAMS a virtual machine twice as expensive is not better in the same magnitude.

Regarding the experimental results in this section, we conclude that there is a very specific point in which is better to use fewer instances to execute an application in the Cloud. That point would depend on the characteristics of the application. In some cases, running the application on instances with lower performance characteristics can be a better option than scaling. This is true when the performance benefits are so minimal that running the application would mean wasting the money.

6. Conclusion

Performance analysis show that Cloud Computing infrastructure offers possibilities to migrate applications from legacy systems and that it is possible to provide an environment for the application while reducing the setup complexity.

Performance variability between experiments was low in Azure, which is an important characteristic for HPC as it leads to a higher predictability of experiments, both in the time it takes and how much it will cost.

The scalability experiments show that it is possible to scale BRAMS up to 168 CPU cores with certain mistrust. Our results show that beyond 88 CPU cores the reduction in execution time is less than two percent at every scaling step.

In the future a comparison with a physical cluster could be performed to improve the soundness of the results. Also, we are going to test performance and scalability using Infiniband equipped instances.

References

[1] R. Souto, R. Avila, P. Navaux, M. Py, N. Maillard, T. Diverio, H. Velho, S. Stephany, A. Preto, J. Panetta, E. Rodrigues, and E. Almeida, "Processing mesoscale climatology in a grid environment," in *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pp. 363–370, May 2007.

[2] C. Evangelinos and C. N. Hill, "Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2," in *In The 1st Workshop on Cloud Computing and its Applications (CCA*, 2008.

[3] S. Benedict, "Performance issues and performance analysis tools for hpc cloud applications: a survey," *Computing*, vol. 95, no. 2, pp. 89–108, 2013.

[4] CPTEC-INPE, "Brazilian Regional Atmospheric Modelling System (BRAMS)." `http://www.cptec.inpe.br/brams`, 2014. Accessed August 10, 2014.

[5] R. Pielke, W. Cotton, R. Walko, C. Tremback, W. Lyons, L. Grasso, M. Nicholls, M. Moran, D. Wesley, T. Lee, and J. Copeland, "A comprehensive meteorological modeling system - RAMS," *Meteorology and Atmospheric Physics*, vol. 49, no. 1-4, pp. 69–91, 1992.

[6] Microsoft, "Sizes for virtual machines." `https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-size-specs/`. Accessed July 1, 2015.

[7] E. Roloff, F. Birck, M. Diener, A. Carissimi, and P. Navaux, "Evaluating high performance computing on the windows azure platform," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 803–810, June 2012.

[8] E. D. Carreño, E. Roloff, and P. O. A. Navaux, "Challenges and Solutions in Executing Numerical Weather Prediction in a Cloud Infrastructure.," in *International Conference on Computational Science*, (Reykjavík, Iceland), p. 6, 2015.