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-

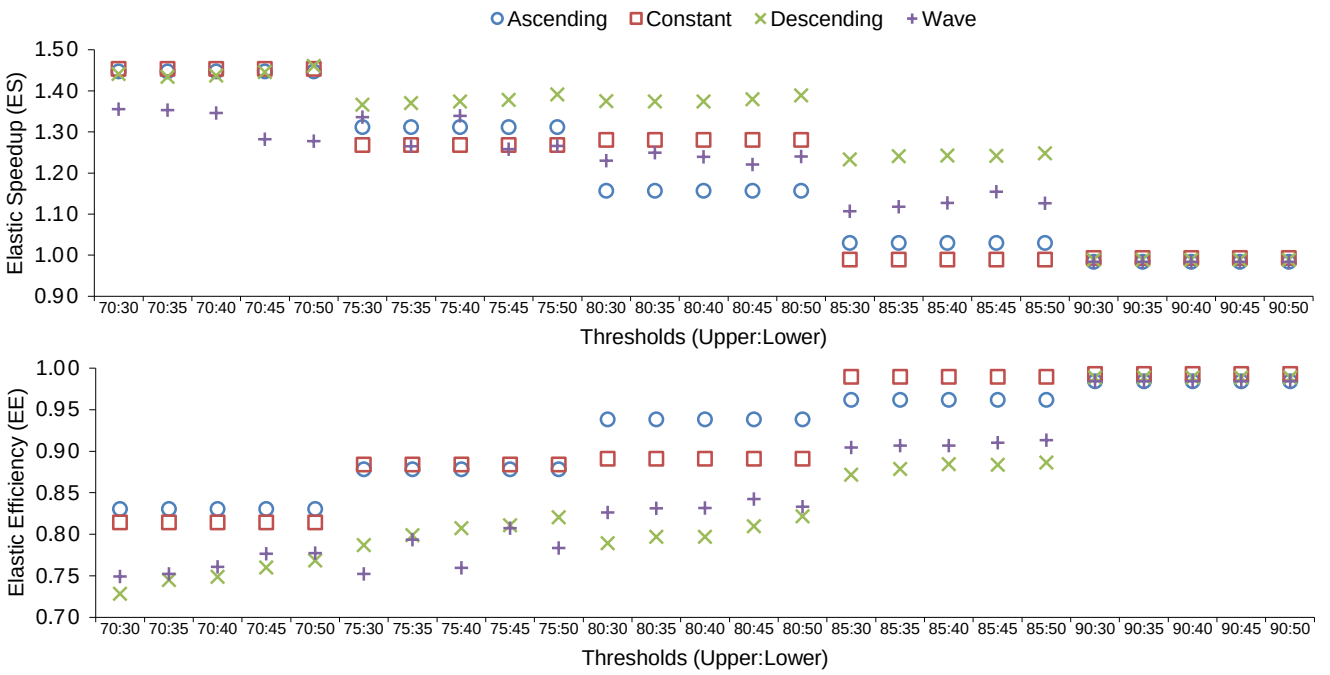

Figure 1. Performance and efficiency results of all application loads in the different scenarios.

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.