Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

Constructing Elastic Scientific Applications Using Elasticity Primitives

Guilherme Galante and Luis Carlos Erpen Bona

Department of Informatics
Federal University of Paraná
Curitiba, PR – Brazil
{ggalante,bona}@inf.ufpr

Abstract. Elasticity can be seen as the ability of a system to increase or decrease the computing resources allocated in a dynamic and on demand way. Considering its importance, some mechanisms to explore elasticity have been proposed by public providers and by academy. However, these solutions are inappropriate to provide elasticity for scientific applications or are limited to a specific programming model. In this context, we present Cloudine, a platform for development of elastic scientific applications based in simple elasticity primitives. These primitives enable the dynamic allocation and deallocation of resources in several levels, ranging from nodes of a virtual cluster, to virtual processors and memory of a node. Using this basic building blocks it is possible to develop applications in different models. The Cloudine effectiveness is demonstrated in the experiments, where two elastic applications in different models were developed.

Keywords: Cloud computing, elasticity, scientific applications.

1 Introduction

Scientific computation is a broad field with applications in many domains of science and industrial settings. Evolution of scientific computation methodology is fast and the construction of infrastructures for computational science lies at the front edge of technical development. During the last decades, scientific applications have been executed over high performance infrastructures, including cluster and grid computing, and more recently cloud computing [1].

Cloud computing refers to a flexible model for on-demand access to a shared pool of configurable computing resources (such as networking, servers, storage, platforms and software) that can be easily provisioned as needed, allowing immediate access to required resources without needing to purchase any additional infrastructure [2]. It implies that the amount of resources used by an application may be changed over time, without any long-term indication about the future demands [3]. This flexibility in resources allocation is called *elasticity*.

Currently, cloud elasticity has been used for scaling traditional web applications in order to handle unpredictable workloads, and enabling companies

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

to avoid the downfalls involved with the fixed provisioning (over and under-provisioning) [4][5]. In scientific scenario, the use of cloud computing is discussed in several studies [6][7], but the use of elasticity in scientific applications is a subject that is starting to receive attention from research groups [8].

This interest is related to the benefits it can provide, that include, improvements in applications performance, cost reduction and better resources utilization. Improvements in the performance of applications can be achieved through dynamic allocation of additional processing, memory, network and storage resources. Examples are the addition of nodes in a master-slave application in order to reduce the execution time, and the dynamic storage space allocation when data exceeds the capacity allocated for the hosted environment in the cloud.

The cost reduction is relevant when using resources from public clouds, since resources could be allocated on demand, instead allocating all of them at the beginning of execution, avoiding over-provisioning. It could be used in applications that use the MapReduce paradigm, where is possible to increase the number of working nodes during the mapping and to scale back resources during the reduction phase. Elastic applications can also increase computational capabilities when cheaper resources became available. An example is the allocation of Amazon Spot Instances, when the price becomes advantageous [9].

In private clouds, the elasticity can contribute to a better resources utilization, since resources released by an user can be instantaneously allocated to another one. It allows an application to start with few resources and increasing them according to availability, without have to wait for the ideal number of resources to become free, as occurs in non-elastic applications.

Elasticity solutions have been proposed by public cloud providers, e.g., Amazon [10], Azure [11] and Rackspace [12], but most of this solutions are suitable for server-based applications, such as, HTTP, e-mail and database servers, which relies on the replication of virtual machines and uses load balancers to distribute the workload among the numerous instances [13]. In general, scientific applications do not adapt to this approach.

Some academic researches have addressed the development of elastic scientific applications. It is possible to find works focusing on workflows [14], MapReduce [9][15], MPI [16] and master-slave applications [17]. However, to the best of our knowledge, there are not frameworks or platforms that could be used independently from a specific application model.

In this context, we present Cloudine, a platform for development and deployment of elastic scientific applications in Infrastructure-as-a-Service (IaaS) clouds. Unlike current elasticity solutions, our platform provides a platform and a set of elasticity primitives for supporting the development of elastic applications in different programming models, e. g., multithread, master-slave, bag-of-tasks, mapreduce and single program multiple data (SPMD) applications.

In addition, Cloudine enables exploration of elasticity in a broad way, allowing (depending on the cloud used) the dynamic allocation and deallocation of resources in several levels, ranging from nodes of a virtual cluster, to virtual processors and memory of a node. The Cloudine effectiveness is demonstrated

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

in the experiments, where two different elastic applications were developed: an OpenMP heat transfer problem and a bag-of-tasks file sorting application.

The remainder of the paper is organized as follows. Section 2 presents an overview of the current cloud elasticity mechanisms. Section 3 introduces the proposed platform. In Section 4 we present an example of how Cloudine works. In Section 5, the experiments are presented and the results are discussed. Finally Section 6, concludes the paper.

2 Cloud Elasticity: An Overview

Many elasticity solutions have been implemented by public cloud providers and by academy [8]. In general, IaaS public cloud providers offer some elasticity feature, from the basic, to more elaborate automatic solutions.

Amazon Web Services [10], one of the most traditional IaaS cloud providers, offers a mechanism based in virtual machine replication called Auto-Scaling. The solution is based on the concept of Auto Scaling Group (ASG), which consists of a set of instances that can be used for an application. Amazon Auto-Scaling uses a reactive approach, in which, for each ASG there is a set of rules that defines the instances number to be added or released. The metric values are provided by CloudWatch monitoring service, and include CPU usage, network traffic, disk reads and writes.

GoGrid [18] and Rackspace [12] also implement replication mechanisms, but unlike Amazon, they do not have native automatic elasticity services. Both providers offer an API to control the amount of virtual machines instantiated, leaving to the user the implementation of more elaborate automated mechanisms.

Other IaaS cloud providers also provide elasticity mechanisms but the features presented are not substantially distinct from presented above. Basically, the current elasticity solutions offer a virtual machine (VM) replication mechanism, accessed using an API or via interfaces, and in some cases, the resources allocation is managed automatically by a reactive (based in rules) controller.

In turn, applications developed in Platform-as-a-Service (PaaS) clouds have implicit elasticity. These clouds provide execution environments, called containers, in which users can execute their applications without having to worry about which resources will be used. In this case, the cloud platform automatically manages the resource allocation, thus, developers do not have to monitor the service status or interact to request more resources [19].

An example of PaaS platform with elasticity support is Google AppEngine [20], a platform for developing scalable web applications (Java, Python, and JRuby) that run on top of Google's server infrastructure. These applications are executed within a container (sandbox) and AppEngine take care of automatically scaling when needed.

Azure [11] is the solution provided by Microsoft for developing scalable .NET applications for clouds. Despite offering platform services, Azure does not provide an automatic elasticity control. In its approach, the user must configure the allocation of resources.

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

Several academic projects also developed elasticity mechanisms. They are similar to those provided by commercial providers, but include new techniques and methodologies for elastic provisioning of resources. Examples of elasticity solutions proposed in academy includes the works of Lim et al. [21], Roy et al. [22], Gong et al. [23], Sharma et al. [24] and several others.

Considering the elasticity solutions presented so far, most of them are appropriate for server-based applications, such as, http, e-mail and database, which relies on the replication of virtual machines and load balancers to distribute the workload among the numerous instances. In general, scientific applications do not adapt to this approach, since most of them need a higher level of coordination, requiring synchronizations and message exchanges. In this context, some works address the development of frameworks and platforms focusing scientific applications.

Aneka [25] is a .NET-based application development platform (PaaS), which offers a runtime environment and a set of APIs that enable developers to build applications by using multiple programming models such as task, thread and MapReduce, which can leverage the compute resources on either public or private clouds. In Aneka, when an application needs more resources, new container instances are executed to handle the demand, using local or public cloud resources.

In the works of Chohan et al. [9] and Iordache et al. [15] the elastic execution of MapReduce applications are presented. In the former, the authors investigate the dynamic addition of Amazon Spot Instances to reduce the application execution time. The latter presents a system that implements an elastic MapReduce API that allows the dynamic allocation of resources of different clouds.

Raveendran et al. [16] present a framework for the development of elastic MPI applications. The authors proposed the adaptation of MPI applications by terminating the execution and restarting a new one using a different number of virtual instances. Vectors and data structures are redistributed and the execution continues from the last iteration. Applications that do not have an iterative loop cannot be adapted by the framework, since it uses the iteration index as execution restarting point.

Rajan et al. [17] presented Work Queue, a framework for the development of elastic master-slave applications. Applications developed using Work Queue allow adding slave replicas at runtime. The slaves are implemented as executable files that can be instantiated by the user on different machines. When executed, the slaves communicate with the master, that on demand coordinates task execution and the data exchange. Based in Work Queue, Yu et al. [26] developed MWPC, an architecture for execution of elastic workflows on heterogeneous distributed computing resources.

Elastic workflows execution is also addressed in the work of Byun et al. [14], where the authors propose an architecture for the automatic execution of large-scale workflow applications on dynamically and elastically provisioned computing resources.

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

How we can observe, each elasticity solution was developed focusing a specific programming model, such as, message-passing (MPI), master-slave, workflow or mapreduce. There is not a generic framework that enables the construction of applications of any programming model. Other point to be observed is that the presented solutions support only horizontal elasticity, i. e., the allocation unit is a virtual machine. Vertical elasticity, in which processing, memory and storage resources can be added/removed from a running virtual instance, is not currently supported by the presented solutions.

In next section, we present our elasticity solution called Cloudine (*Cloud Engine*), whose objective is to provide mechanisms to the development of elastic applications in different programming models and supporting exploration of horizontal and vertical elasticity.

3 Cloudine: Elasticity via Programming Primitives

Cloudine is a platform for development and deployment of elastic scientific applications in computational clouds. The platform focuses parallel and distributed non-interactive batch applications that runs directly over the VM's operating system (IaaS clouds). Unlike the solutions proposed in Section 2, which are restrict to a programming model, our solution is based in low level elasticity primitives, with which is possible to construct applications in different models, according to the program structure.

These primitives enable the dynamic allocation and deallocation of resources in several levels, ranging from nodes of a virtual cluster, to virtual processors and memory of a node. Primitives for collecting information and monitoring data from the cloud system is also provided. Thus, the platform enables the development of elastic applications, by inserting basic building blocks into the source code, letting the application itself to control its elasticity.

In addition, Cloudine provides mechanisms to deployment and execution of applications using cloud resources in an automatic and easy way. The platform manages the creation of the virtual environment and coordinates the entire execution process, from source code submission to results collection phase.

To enable the development and deployment of elastic applications, Cloudine utilizes three main components, *Resources and Execution Manager*, *User Interface* and *Elasticity API*, as shown in Fig. 1, and described in next sections.

3.1 Resources and Execution Manager

The *Resources and Execution Manager* (REM) is the module that coordinates the applications execution and controls all resources allocation. This module is composed by five components, as presented in Fig. 2.

The *REM Server*, receives and analyses the requests sent by the other Cloudine modules and dispatches them to the appropriate component. These requests include the allocation of new virtual environments, allocation of complementary

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

resources, deallocation of resources, status information and execution of applications. REM is also responsible for sending responses and error codes to other modules.

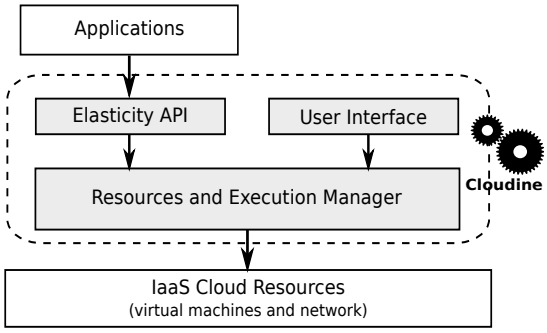

Fig. 1. Cloudine architecture

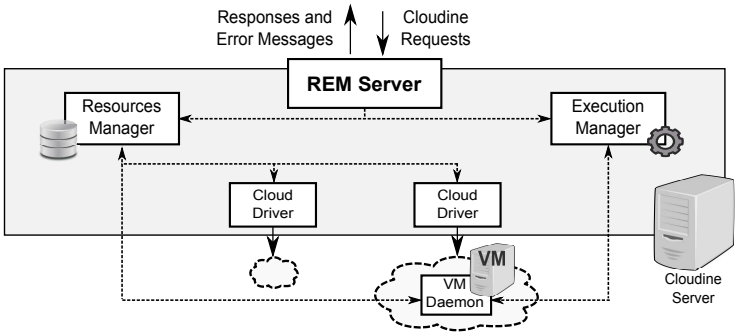

Fig. 2. Resources and Execution Manager modules

All requests involving resources are handled by the *Resources Manager*. It receives from REM Server the demands and allocates through *Cloud Drivers* the requested resources. Cloud Drivers are used to translate the Cloudine requests to a set of commands of a specific cloud, allowing the use of several providers since a driver is implemented for each one of them. Thus, it is possible to use a single interface internally, independently from the cloud used. To date, it is not possible to combine resources from different clouds.

The execution requests are processed by *Execution Manager*. This module is responsible for upload the application dependencies, build and execute the application. To perform these tasks on each VM, a *VM Daemon* is used. A VM Daemon is automatically started at the VM initialization and its role is to run the

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

actions submitted by Execution Manager (e.g., copy file, compile, execute script) internally in the virtual machine, and collect the application status. VM Daemon is also used by Resources Manager to obtain the VM resources information.

3.2 User Interface

Cloudine provides a command-line *User Interface* (UI) that is used to request the initial virtual environment and to submit the applications. This module is also used to track the execution progress and receive error messages.

The virtual environment is configured in a file containing a set of attribute-value, that can vary according to the cloud (or driver) used. The attributes can include number of nodes, number of Virtual CPUs (VCPU), memory amount, virtual machine image or instance type, networking configurations and others. These settings are used to create a template that is sent to REM, which provides the virtual environment creation.

Similarly, the application execution is described by an attribute set that comprises script name, the applications dependencies (files to be uploaded) and the output directory, where results will be saved. The execution is requested to REM via command-line, which coordinates the process. An example of configuration file is presented in Fig. 3.

```
[resources]
cloud=OpenNebula
env_type=cluster
nodes=4
vcpu=2
memory=512
image=ubuntu.img

[app]
name=test
exec=exec.sh
exec_on_node=1
work_dir=/home/guilherme/test
```

Fig. 3. Example of a simple configuration file

In this example, a cluster of four nodes is requested in an OpenNebula cloud. Each node (VM) has two virtual CPUs (vcpu), 512 MB RAM, and uses an image called *ubuntu.img*. The rest of the file describes the application name, the script name (automatically executed on node 1), and the application work directory.

3.3 Elasticity API

The *Elasticity API* provides a set of primitives that enable the implementing of elastic applications for the Cloudine platform. The use of an API to develop

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

elastic applications is interesting because the elasticity control is done by the application itself, which relies on internal information and monitoring data coming from the platform to request its own resources and does not require external mechanisms or user interaction.

Besides the creation of elastic applications, another possible use of the API is the construction or adaptation of parallel processing middleware that transparently supports elasticity. This feature could be used, for example, to adjust the processing capability when occur changes in the number of threads in an OpenMP application through `omp_set_num_threads()` function. A second example is the creation of a new virtual machines when a new MPI process is created with the `MPI_Comm_spawn()` primitive, defined in MPI-2 standard.

To date, the API supports C/C++ languages and offers 12 primitives, providing dynamic allocation of VCPUs, memory and virtual machines. Information about CPU and memory is also provided. As required, new primitives can be added in the future. Table 1 shows the functions implemented so far and their descriptions.

Table 1. Cloudine API functions

Function	**Description**
`int clne_add_vcpu(int N)`	Add N VCPUs to the current VM. Return 1 if successful, 0 if not.
`int clne_rem_vcpu(int N)`	Remove N VCPUs from the current VM. Return 1 if successful, 0 if not.
`int clne_add_node(int N)`	Add N nodes to the virtual environment (cluster). Return 1 if successful, 0 if not. This function also creates (or updates) a file in the VM containing the IPs of the cluster machines.
`int clne_rem_node()`	Turn off and remove the actual node from the virtual environment (cluster). Return 1 if successful, 0 if not.
`int clne_add_memory(long int N)`	Add N megabytes of memory to the current VM. Return 1 if successful, 0 if not.
`int clne_rem_memory(long int N)`	Remove N megabytes of memory from the current VM. Return 1 if successful, 0 if not.
`int clne_get_freemem()`	Returns the free memory amount of the VM host machine.
`int clne_get_maxmem()`	Returns the total memory amount of the VM host machine.
`int clne_get_mem()`	Returns the total memory amount of the current VM.
`int clne_get_freecpu()`	Returns the free CPU amount of the VM host machine.
`int clne_get_maxcpu()`	Returns the total CPU amount of the VM host machine.
`int clne_get_vcpus()`	Returns the CPU amount of the current VM.

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

To use the API the user must include the Cloudine library `clne.h` and compile passing the "`-lclne`" flag to compiler. The files `clne.h` and `libclne.so` are automatically provided in the VMs instantiated by Cloudine.

All primitives are implemented in `libclne.so`, that makes the requests to REM. When a primitive is invoked, a message including the operation and the arguments is assembled and sent to Resources Manager via TCP sockets. Resources Manager sends the request to Cloud Driver, that executes the operation in the cloud. According to operation result, a return code is sent back to application. Thus, depending on the cloud used or the operation, the response time can vary from few milliseconds (e.g., get operations) to minutes (VM request).

It is important to highlight that not all clouds support all functions. For example, Amazon EC2 does not support the dynamic allocation of memory or VCPUs, supporting only the allocation and deallocation of virtual machine instances. The information about the supported functions must be implemented in the corresponding Cloud Driver.

4 Cloudine Operation

To demonstrate how Cloudine works, this section presents an example of using the platform to instantiate a virtual machine and to run an elastic application. The process is illustrated in 14 steps as shown in Fig. 4 and explained in sequence.

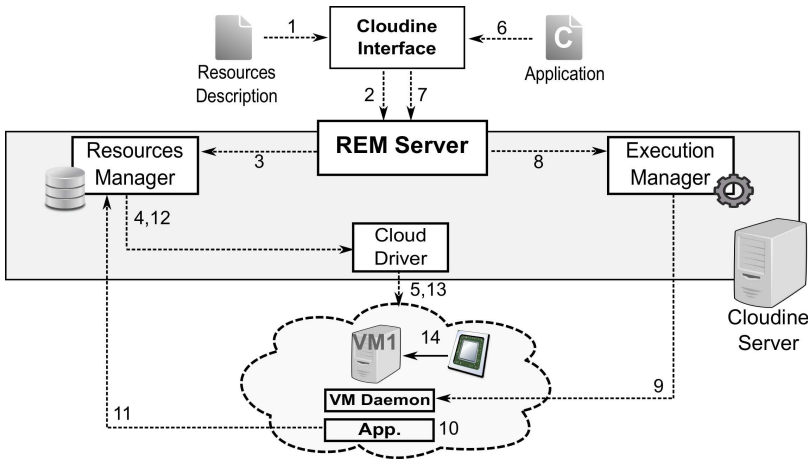

Fig. 4. Resources allocation and application execution steps

The process starts when (1) the user request a virtual machine via Cloudine Interface; REM Server (2) receives and (3) forwards the request to the Resource Manager that (4) chooses the correct Cloud Driver and sends an allocation command. (5) A new virtual machine (VM1, in the figure) is created in the cloud.

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

After the VM creation, (6) the application is submitted to execution, that is handled by (7) REM Server and (8) Execution Manager. (9) Execution Manager uses the VM Daemon to (10) start the elastic application on the VM1.

To exemplify the use of elasticity API, suppose that application request a new VCPU. This request is (11) sent to Resources Manager that (12) asks to Cloud Driver if this operation is supported by the cloud. If the allocation is possible, (13) the Cloud Driver sends the command to the cloud. Finally, (14) the VCPU is added to the VM. Note that the communication between application and Resource Manager is provided by the Cloudine API.

5 Tests and Results

In order to evaluate the proposed platform, a prototype was implemented using Python, C and Shell Script. An OpenNebula cloud [27] (version 3.4) using Xen as virtualization technology was used to perform the experiments. The choice of a private OpenNebula cloud was made based on the flexibility for VMs creation, which allows the configuration of resources according to application needs, unlike Eucalyptus and OpenStack, in which a pre-configured instance class must be chosen. The Xen Hypervisor was used due its capacity to handle the allocations of VCPUs and memory on-the fly, supporting all Cloudine API primitives presented in Table 1.

The hardware used is a 24-core workstation equipped with 3 AMD Opteron 6136 at 2.40 GHz and 96 GB RAM. We used 4 cores and 4 GB RAM for dom0 and the remaining CPUs and memory for the other VMs. The operating system in all tests, including virtual machines, is Ubuntu Server 12.04 with kernel 3.2.0-29.

To validate our proposal, we performed two tests using applications developed in distinct programming models. In the first test, we evaluate the elasticity in fine-grain level, adding dynamically new VCPUs to an existing VM running an OpenMP heat transfer problem. In the second test, we evaluate the elasticity in coarse-grain level, using the node creation primitives in a bag-of-task file sorting application.

5.1 OpenMP 2D Heat Transfer Problem

The heat transfer problem consists in solving a partial differential equation to determine the variation of the temperature within the heat conducting body. The code is implemented in C and uses OpenMP to provide the threads parallelism.

The application has an iterative loop that determines the temporal evolution of the simulation. At the beginning of the loop, a `clne_get_freecpu()` primitive was inserted in the source code to verify if there are idle CPUs. If CPUs were available, they are allocated to the VM using the `clne_add_vcpu()` primitive. The number of active OpenMP threads is set to the number of VCPUs of the VM.

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

In Fig. 5 is presented the results of the elastic allocation of resources. In the beginning of the simulation, the VM has only 2 VCPUs and 10 GB RAM. In this experiment, every 240 seconds 2 new VCPUs are available to allocation, as shown in Fig. 5.

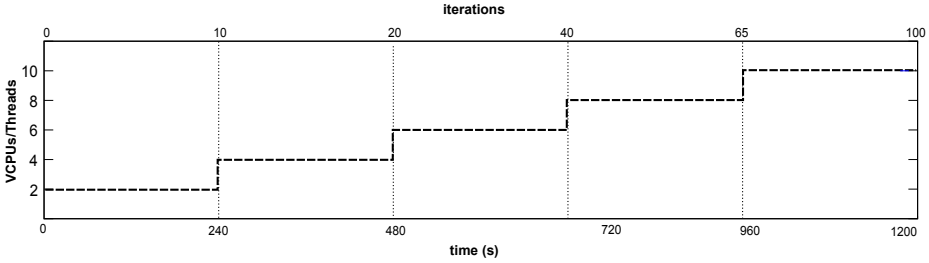

Fig. 5. Elastic allocations of VCPUs

Figure 6 shows the execution time and the speedup of normal and elastic versions. The average execution time (1,230 seconds) and the speedup (5.2) obtained by the elastic version are very close to those obtained by the normal version using 6 threads. The overhead caused by the insertion of the primitives is 4.6% or 56 seconds.

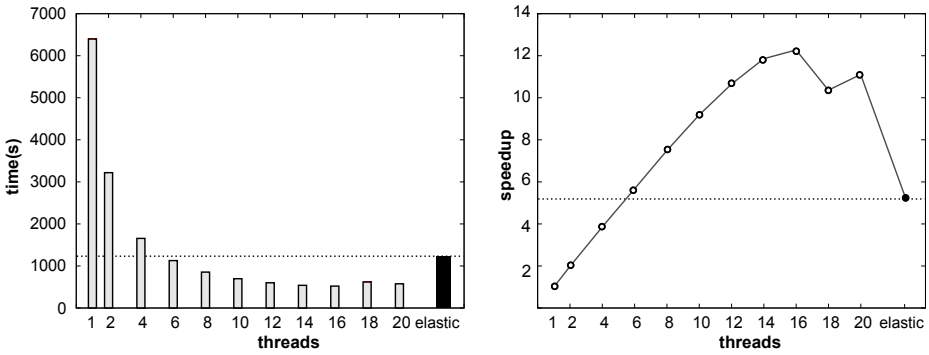

Fig. 6. Execution time and speedup

It is important to note that the inelastic version in the same scenario (with only 2 VCPUs available at the starting time) present average execution time of 3,220 seconds, i. e., 2.6 times slower than the elastic version. It is clear that the performance gains may vary according to the amount of available resources during execution, achieving more significant results when resources become available early or in greater quantities.

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

5.2 Bag-of-task Files Sorting

In second experiment, we tested a file sorting application implemented in bag-of-task model. The application consists of one server and N workers. The role of the server is to control the list of files to be sorted and to create dynamically the workers. The workers only sort the files.

Initially, the server is started and immediately instantiates a new VM to host a worker using `clne_add_node()`. Next, the worker application is upload and started in VM. The worker communicates with the server to get a file to sort, download the file and perform the operation. New workers are created (if there are available resources) while there are files to be sorted. At execution's end, all worker nodes are deallocated using the `clne_rem_node()` primitive.

In the tests, we used 400 files with 100,000 lines each. A timeline showing the workers allocation is presented in Fig. 7. The white balls represent the allocation request sent by the server and the black balls represent the moment when the worker operation is started.

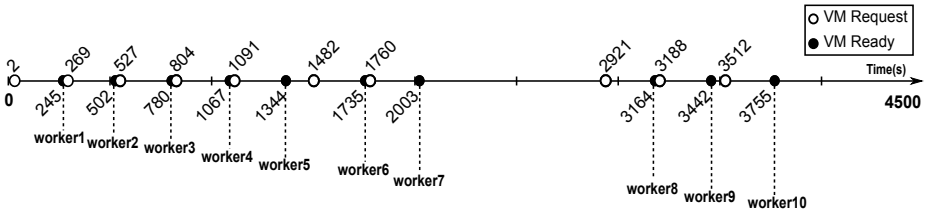

Fig. 7. Dynamic allocation of workers

At the beginning of the execution, 7 workers are sequentially allocated, taking advantage of available resources. After 900 seconds, new resources became available and the last three workers are allocated. The experiment ends at approximately 4,500 seconds, using 10 workers. The VM allocation time is on average 4 minutes.

As in the previous experiment, the platform proved to be efficient in allocating resources elastically, enabling the use of idle resources of the private cloud, and consequently improving application performance.

6 Conclusion and Future Work

In this paper we presented a platform for development and deployment of elastic scientific applications in IaaS clouds. The main contributions of our platform is the exploration of the elasticity using simple primitives, allowing the development of applications in several paradigms and models, and exploring horizontal and vertical elasticity.

The results show that Cloudine was successfully used to provide elasticity for different applications, being used to reduce the execution time and to increase

Published in Proceedings of 13th International Conference on Computational Science and Applications, 2013. v. 5. p. 281-294.

the utilization of shared resources in a private cloud. Besides, the platform could be also employed in other programming models and scenarios, depending on the application and goals to be achieved.

As future work, we plan to develop elastic parallel middleware (e. g. Elastic OpenMP) with which users will be able to construct applications transparently, or to port applications with few modifications in source code. We also intend to develop cloud drivers to support different clouds, such as, OpenStack, Eucalyptus and public clouds.

Acknowledgments. This work is supported by CAPES and INCT-MACC (CNPq grant nr. 573710/2008-2).

References

1. Villamizar, M., Castro, H., Mendez, D.: E-Clouds: A SaaS Marketplace for Scientific Computing. In: 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing. UCC 2012, pp. 13–20. IEEE (2012)
2. Badger, L., Patt-Corner, R., Voas, J.: Cloud Computing Synopsis and Recommendations Recommendations of the National Institute of Standards and Technology. Technical report, NIST Special Publication 146 (2011)
3. Leymann, F.: Cloud Computing: The Next Revolution in IT. In: 52th Photogrammetric Week, Wichmann Verlag. pp. 3–12. (2009)
4. Chieu, T.C., Mohindra, A., Karve, A.A., Segal, A.: Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. In: 2009 IEEE International Conference on e-Business Engineering. ICEBE 2009, pp. 281–286. IEEE (2009)
5. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, a., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, Ionaharia, M.: A View of Cloud Computing. Commun. ACM 53 vol.4, 50–58 (2010)
6. Wang, L., Zhan, J., Shi, W., Liang, Y.: In Cloud, Can Scientific Communities Benefit from the Economies of Scale?. IEEE Transactions on Parallel and Distributed Systems 23 vol. 2, 296–303 (2012)
7. Oliveira, D., Baio, F.A., Mattoso, M.: Migrating Scientific Experiments to the Cloud. HPC in the Cloud. `http://www.hpcinthecloud.com/hpccloud/2011-03-04/migrating_scientific_experiments_to_the_cloud.html`
8. Galante, G., Bona, L.C.E.: A Survey on Cloud Computing Elasticity. In: International Workshop on Clouds and eScience Applications Management. CloudAM 2012, pp. 263–270. IEEE/ACM (2012)
9. Chohan, N., Castillo, C., Spreitzer, M., Steinder, M., Tantawi, A., Krintz, C.: See Spot Run: Using Spot Instances for Mapreduce Workflows.In: 2nd USENIX Conference on Hot topics in Cloud Computing. HotCloud 2010, USENIX Association (2010)
10. Amazon Web Services, `http://aws.amazon.com/`
11. Microsoft Azure, `http://www.windowsazure.com/`
12. Rackspace, `http://www.rackspace.com/`
13. Vaquero, L.M., Rodero-Merino, L., Buyya, R.: Dynamically Scaling Applications in the Cloud. SIGCOMM Comput. Commun. Rev. 41, 45–52 (2011)
14. Byun, E.K., Kee, Y.S., Kim, J.S., Maeng, S.: Cost Optimized Provisioning of Elastic Resources for Application Workflows. Future Gener. Comput. Syst. 27 vol. 8, 1011–1026 (2011)

15. Iordache, A., Morin, C., Parlavantzas, N., Riteau, P.: Resilin: Elastic MapReduce over Multiple Clouds. Technical report RR-8081, INRIA (2012)
16. Raveendran, A., Bicer, T., Agrawal, G.: A Framework for Elastic Execution of Existing MPI Programs. In: International Symposium on Parallel and Distributed Processing Workshops and PhD Forum. IPDPSW 2011, pp. 940–947. IEEE (2011)
17. Rajan, D., Canino, A., Izaguirre, J.A., Thain, D.: Converting a High Performance Application to an Elastic Cloud Application. In: 3rd International Conference on Cloud Computing Technology and Science. CLOUDCOM 2011, pp. 383–390. IEEE (2011)
18. GoGrid, `http://www.gogrid.com/`
19. Caron, E., Rodero-Merino, L., F. Desprez, A.M.: Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds. Technical report RR-7857, INRIA (2012)
20. Google App Engine, `http://code.google.com/appengine`
21. Lim, H.C., Babu, S., Chase, J.S., Parekh, S.S.: Automated Control in Cloud Computing: Challenges and Opportunities. In: 1st Workshop on Automated Control for Datacenters and Clouds. ACDC 2009, pp. 13–18. ACM (2009)
22. Roy, N., Dubey, A., Gokhale, A.: Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In: 4th International Conference on Cloud Computing. CLOUD 2011, pp. 500–507. IEEE (2011)
23. Gong, Z., Gu, X., Wilkes, J.: PRESS: PRedictive Elastic ReSource Scaling for Cloud Systems. In: 6th International Conference on Network and Service Management. CNSM 2010, pp. 9–16. IEEE (2010)
24. Sharma, U., Shenoy, P., Sahu, S., Shaikh, A.: A Cost-Aware Elasticity Provisioning System for the Cloud. In: 31st International Conference on Distributed Computing Systems. ICDCS 2011, pp. 559–570. IEEE (2011)
25. Calheiros, R.N., Vecchiola, C., Karunamoorthy, D., Buyya, R.: The Aneka Platform and QoS-Driven Resource Provisioning for Elastic Applications on Hybrid Clouds. Future Generation Computer Systems 28(6), 861–870 (2011)
26. Yu, L., Thain, D.: Resource Management for Elastic Cloud Workflows. In: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. CCGRID 2012, pp. 775–780. IEEE (2012)
27. OpenNebula, `http://www.opennebula.org/`