# ERHA: Execution and Resources Homogenization Architecture

Guilherme Galante, Luis Carlos Erpen de Bona
Department of Informatics
Federal University of Paraná
Curitiba, PR – Brazil
{ggalante,bona}@inf.ufpr.br

Paulo Antonio Leal Rego, José Neuman de Souza
Master and Doctorate in Computer Science
Federal University of Ceará
Fortaleza, CE – Brazil
pauloalr@lia.ufc.br, neuman@ufc.br

***Abstract*—In this paper, we present the Execution and Resources Homogenization Architecture (ERHA). The architecture aims to provide mechanisms for submitting and executing batch applications in private IaaS clouds using homogeneous virtual environments created over heterogeneous physical infrastructure. With ERHA it is possible to deploy and execute applications in IaaS clouds in an automatic and easy way. The architecture creates homogeneous virtual environments and manages the entire execution process, from source code submission to results collection phase. The results confirmed the architecture efficiency in deploying parallel applications in clouds and reducing significantly the disparities in execution time using different machine models.**

***Index Terms*—scientific applications; application execution; homogeneous environments.**

## I. Introduction

The rapid provisioning of independent and isolated resources, hardware and software customization, quick access to resources as well as on-demand scalability, have made cloud computing an attractive model for the scientific community. In fact, many scientists have adopted this new paradigm, moving their data and performing *in silico* experiments in the cloud [1]–[4].

However, the deployment and execution of scientific applications are generally not straightforward, comprising a set of complex hardware and software configurations. Other issues must be considered in a cloud scenario, e.g., particular control aspects of different clouds, interfaces to be used, number of virtual machines (VMs) to be deployed and what resources will be needed by them.

According to [5], running a scientific application in the cloud presents three different challenges: initial application deployment, subsequent application execution, and data transfers to/from the cloud. Generally, the application deployment in a cloud requires that all software (and possibly data) be stored in a VM image, which is sent to and stored in the cloud. Thus, the user can create a new VM from this image, access it and run applications. At the end, the application results must be copied to a non-volatile storage using a network protocol. These steps seem extremely simple for a computer scientist, but may pose a challenge to people from other areas. The challenge may be even greater when it comes to parallel applications.

Another related problem is the provisioning of computing resources in heterogeneous physical infrastructures. The VMs performance is directly related to the physical machines (PMs) where they are allocated. As the cloud computing environment is commonly highly heterogeneous, there may be machines with different processors. This difference between the CPUs can directly influence the performance of VMs and consequently the applications encapsulated within. An example is presented in [6], where the performance results of a MapReduce execution on Amazon EC2 show fluctuations up to 24%. Another issue related to performance disparities due to heterogeneity is that the slowest instance will be the bottleneck for an entire execution of the application, thus, under-utilizing faster machines.

Considering these issues, this paper presents the Execution and Resources Homogenization Architecture (ERHA), a solution to automate the deployment and execution of sequential and parallel batch applications in clouds, providing homogeneous computing resources. This type of application was focused because it is very common in scientific computations. The architecture provides a language to describe the resources and the execution parameters, a standard method to deploy and execute the application in clouds, and a mechanism that enables the allocation of VMs using processing units (PUs), a metric that represents the effective processing power of each machine (physical or virtual).

Four experiments show the architecture efficiency in deploying applications in clouds in an easy way and reducing significantly the disparities in execution time using different PMs models.

The remainder of the paper is organized as follows. Section II presents the related works. Section III introduces the proposed architecture. In Section IV, the experiments are presented and the results are discussed. Finally, in Section V, we present our conclusion and potential future research topics.

## II. Related Work

Several studies [2], [7], [8] have assessed the aspects of running scientific applications on public clouds. The feasibility of the current cloud services (Amazon and GoGrid) for the execution of scientific applications is explored in [7]. A performance comparison of eight applications (CAM, Gamess, GTC,

 ISBN: 978-1-61208-216-5 

IMPACT-T, MAESTRO, MILC, Paratec and HPCC) running in a private virtual cluster and in Amazon EC2 is presented in [8]. In a similar study, the impact of using different MPI libraries in an atmospheric model running on EC2 is analyzed in [2].

All these studies present some concerns about applications performance. Large fluctuations of high-performance computing workloads on cloud infrastructure were reported in [9]. In [7], the authors have found that many Amazon and Google services exhibit large performance changes over time.

The studies which use techniques for limiting the CPU usage of VMs are generally related to the dynamic provisioning of resources. In [10], Xen's performance isolation for I/O intensive applications is studied and two mechanisms are proposed to improve CPU and network resource isolation. In [11], an architecture for dynamic resources management upon the hypervisor Xen is presented. The authors dynamically adjust the amount of CPU and memory of VMs to reduce service level objectives (SLOs) violation. In [12], an adaptive control system which automates the task of tuning the resources allocation for the maintenance of SLOs is presented. The work uses KVM hypervisor and focuses on maintaining the expected response time for Web applications, by tuning the CPU usage.

Unlike aforementioned works, the hypervisors KVM and Xen are used in ERHA and the techniques for limiting the CPU usage are leveraged to handle the problem of providing homogeneous resources despite being in a heterogeneous infrastructure. In addition, besides Amazon EC2 (with the Elastic Compute Unit), no other public cloud provider or research uses an abstraction like our PU for representing CPU resources.

About the creation of virtual environments and the execution of applications, the three most similar works are highlighted: Neptune [13], DADL [14] and Nimbus Context Broker [15]. Neptune is a domain specific language that automates configuration and deployment of existing HPC software in AppScale clouds. DADL (Distributed Application Description Language) is a language for describing hardware requirements, behavior and architecture of distributed applications. Finally, Nimbus Context Broker is a tool used to create and provide virtual clusters.

Cloud computing is taking larger proportions in the scientific field and there are several studies about scientific applications performance in cloud. Despite this, to the best of our knowledge, there is no published research addressing the support for running sequential and parallel batch applications in clouds, especially considering the virtual environments homogenization, as we show with ERHA.

## III. ERHA Architecture

ERHA is an architecture that aims to provide mechanisms to submit and execute batch applications in private IaaS clouds (e.g., OpenNebula and Eucalyptus) using homogeneous virtual environments created over heterogeneous physical infrastructure. With ERHA it is possible to deploy and execute applications in clouds in an automatic and easy way. The architecture creates the virtual environment and manages the entire execution process, from source code submission to results collection phase.

Furthermore, the architecture allows users to configure their VMs' processing power using an uniform metric, the Processing Units (PU). A PU is a value in terms of GFLOPS (billion floating-point operations per second), or other metric that represents the processing power of each physical or virtual machine. The system administrator defines a value to the PU, which is used in the VMs allocations. All virtual instances requested by the user will have equivalent computing power, regardless the underlying PM and its processor model. For example, considering a PM with 30 GFLOPS and the PU set as 10 GFLOPS, it is possible to allocate three VMs with 1 PU, or one VM with 1 PU and another with 2 PUs.

To implement the execution environment and the resources homogenization, ERHA uses three layers, as shown in Figure 1: (1) *Resource Description Layer*, (2) *Execution Management Layer* and (3) *Allocation Layer*.

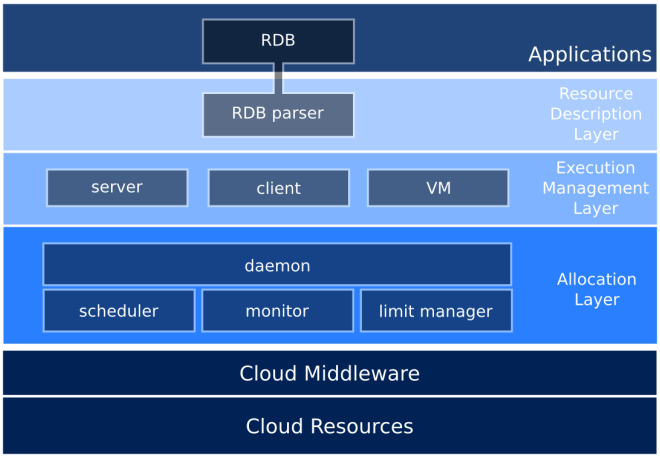



Fig. 1. Architecture layers

The *Resource Description Layer*, *Execution Management Layer* and *Allocation Layer* are presented in Section III-A, III-B, III-C, respectively. A complete example of running an application envolving all ERHA layers is presented in Section III-D.

### A. Resource Description Layer

The *Resource Description Layer* (RDL) is responsible for receiving the resources demands and informing it to the other layers. The resources needed by the application are informed in the *Resource Description Block* (RDB). The block is made up of a set of attribute-value pairs and it must be inserted into the source code, marked by the reserved word *#neb_config* and delimited by braces ({}). An example of RDB is shown in Figure 2.

In this example, a VM named OMPTest is requested in an OpenNebula cloud. The VM has four virtual CPUs (VCPU), eight PUs and 256 MB RAM, no disk attachment is needed

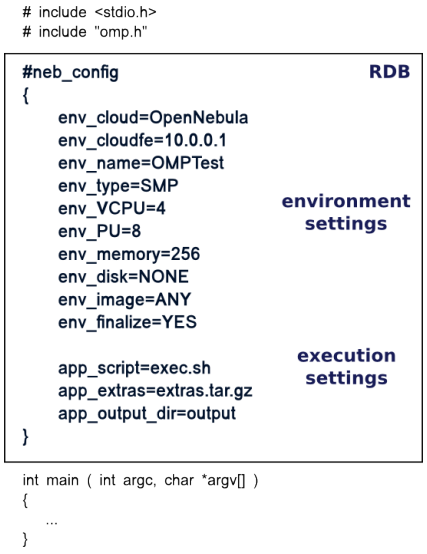



Fig. 2. RDB example

and a default image is used. When the execution ends, the VM will be finalized. Note that in RDB is informed the total amount of PUs of a VM, and this processing power is divided by all VCPUs (in this example, each VCPU has 2 PUs).

The rest of RDB describes the execution script name, the *extras.tar.gz* file, which contains the applications dependencies, and the output directory, where results will be saved.

The information is obtained by the *RDB Parser*, which parses all fields and converts them to the specific cloud middleware format (OpenNebula, in this example). The parser sends the formatted data to the *Execution Management Layer*, which will use them to request the resources to the cloud via *Allocation Layer*.

### B. Execution Management Layer

The *Execution Management Layer* (EML) handles the execution process, controlling all needed interactions with the cloud during the application execution. EML is a client-server application composed of three components: 1) *Client*, 2) *Server* and 3) *VM-Daemon*.

The *Client* is installed in the user machine and provides a command-line interface that is used to submit the applications to the cloud. This interface is also used to track the execution progress and receive error messages. It is also responsible for receiving the resources description from *RDB Parser* and sending it to *Server* module, deploying the application in the cloud, managing the execution and collecting the results.

The *Server* runs in the cloud front-end and is responsible for receiving the requests and sending the performing the actions related to VMs creation and finalization. It receives the resources description from the *Client*, converts this information to the appropriate format and makes the requests to the *Allocation Layer*. The same process is performed to finalize the virtual environment. It is necessary just one *Server* instance to serve all *Clients*.

The last component is the *VM-Daemon*. This module must be inserted into VM image and is initialized on VM boot process. When started, the *VM-Daemon* sends and receives information to/from the *Client*, including IP address, VM identification, authorization keys, error warnings and commands that must be executed in VM.

### C. Allocation Layer

The *Allocation Layer* (AL) was designed to perform the VMs allocation based on PUs. AL ensures uniform allocation of computing power to VMs and standardizes the representation of the processing power of a cloud infrastructure through the use of PUs. The PU is the abstraction used for representing the processing power, similar to ECU (Elastic Computing Unit) for Amazon EC2.

Despite the VM performance being directly related to the underlying PM, the use of PUs metric makes it possible to guarantee VM processing power without worrying about the infrastructure heterogeneity. Four modules make up the AL: (1) *Limit Manager*, (2) *Monitor*, (3) *Scheduler* and (4) *Daemon*.

*Limit Manager* is the module responsible for applying the limits on CPU usage. When a new VM is created, for example, the *Daemon* invokes the *Limit Manager*, which sets the CPU resource that can be used by the VM, considering the amount of PUs allocated to it.

This feature is implemented limiting the CPU cycles used by each VM. To achieve the desired results, it is necessary that the hypervisors allow a way to set the percentage of CPU that will be used by a given VM. Xen has this functionality natively implemented through Credit Scheduler algorithm [16]. An alternative for limiting the CPU usage for the Kernel-based Virtual Machine (KVM) with the *cpulimit* tool is evaluated in [17] and is used in this work.

*Monitor* is the module responsible for capturing information about the entire infrastructure. This module can directly access the infrastructure data or make requests to the cloud computing middleware. The monitored information is related to CPU, memory, storage and network for each PM and VM.

All decisions about the VMs allocation on PMs are taken by *Scheduler*. Unlike the schedulers present in the main cloud computing middleware, like Eucalyptus and OpenNebula, the proposed scheduler must not consider the raw information about CPU, memory and hard drive to define where the VMs will be allocated. Once the architecture is based on PUs, all allocations must be made based on the number of PUs required by the VM and the amount of free PUs in the PMs. The architecture provides some basic scheduling policies, e.g., Random, Round Robin and First Fit. New scheduling policies can be easily created and added to the architecture.

The last module, the *Daemon*, manages all the other modules and controls the infrastructure. It communicates with the cloud computing middleware and maintains information about the PMs and running VMs (captured by the Monitor). In addition, it has access to all the requests that come to the middleware. The *Daemon* receives the requests from

 ISBN: 978-1-61208-216-5 

*Execution Layer* and invokes the *Scheduler* when there are VMs at pending state. When the *Daemon* receives the VM-PM mapping from *Scheduler*, it allocates the VMs. The *Daemon* is also responsible for storing information about the PUs and invoking the *Limit Manager* to apply the limit of CPU usage when a VM is started or migrated.

### D. Running Applications with ERHA

To demonstrate how ERHA works, this section presents an example of using the architecture to run the scenario described in the RDB shown in Figure 2. The environment creation and the application execution is performed in thirteen steps, as illustrated in Figure 3 and explained in sequence.

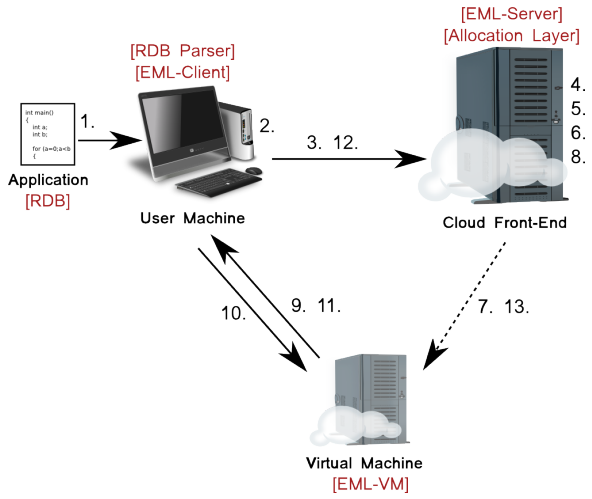



Fig. 3. Application Execution Steps

The process starts when (1) the user adds the *RDB* to the application, provides the scripts and other dependencies and submits the application through *EML-Client*. The *Client* (2) uses the RDB Parser to get information about the requested environment and (3) sends the VM template to *EML-Server*, which (4) requests a VM to the *Allocation Layer*. In the *Allocation Layer*, (5) the *Daemon* detects a pendency and calls the *Scheduler*, that (6) returns a VM-PM mapping. Then, (7) the *Daemon* sends the VM creation request to the cloud middleware, and the VM is created. Next, the *Daemon* calls the *Limit Manager* to (8) apply the CPU limitation to the VM, based on the PU configuration established in RDB.

After VM operating system booting process, (9) the *EML-VM* gets the IP and the VM public-key and sends them to *EML-Client*. The IPs are used to identify the VMs during the execution process, and the public-key is used to allow passwordless remote access. In the following step, (10) *EML-Client* uploads the application and its dependencies to the VM and sends the compilation and execution commands. After finishing the application execution, the (11) *EML-VM* collects the results and sends them back to the *Client*. If the machines are no longer needed, (12) *EML-Client* requests the VM destruction to the *Allocation Layer* and (13) the *Daemon* sends the command for VM destruction to the cloud middleware.

## IV. Tests and Results

To evaluate the proposed architecture, a prototype was implemented using Python, Ruby and Shell Script. OpenNebula 2.2.1 was used as cloud middleware and KVM and Xen as virtualization technology. The choice of OpenNebula 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 class must be chosen.

TABLE I
Physical Machines Configurations

| | Ci7 | Ci5 | X4 | C2D |
|---|---|---|---|---|
| **Processor** | Intel Core i7-930 2.8 GHz | Intel Core i5-750 2.66 GHz | Intel Xeon X3430 2.4 GHz | Intel Core 2 Duo E7400 2.8 GHz |
| **PUs (Total)** | 10 | 9 | 8 | 5 |
| **Memory** | 24 GB | 4 GB | 8 GB | 4 GB |
| **OS** | Ubuntu Server 11.04 64 bits | | | Debian 6.0 64 bits |
| **Hypervisor** | KVM | | | Xen |

A heterogeneous private cloud with OpenNebula was used as testbed, consisting of 2 machines model Ci5, 1 machine model Ci7, 1 machine model X4 and 1 machine model C2D, all connected by a Gigabit Ethernet network. The configurations of the physical machines are presented in Table I. All VMs use Ubuntu Server 11.04 32 bits as operating system. All source code and applications used in experiments are available in the project site [18].

The Intel Linpack benchmark was used to collect the VMs' computing power running inside all PMs. With this information, it is possible to find the relation between CPU usage (%) and the amount of PUs for all PMs. For this work, the PU was set to 3 GFLOPS (50% of one core of X4), and the relation %CPU/PU is presented in Table II. This specific configuration is more suitable for CPU-intensive applications because only the Linpack benchmark is used in the process of PU definition.

TABLE II
Relation between the CPU usage and the amount of PUs for each physical machine.

| | CPU usage | | | |
|---|---|---|---|---|
| | **Ci7** | **Ci5** | **X4** | **C2D** |
| **1 PU** | 39% | 41% | 50% | 44% |
| **2 PUs** | 78% | 81% | 100% | 83% |
| **3 PUs** | 119% | 118% | 150% | 130% |
| **4 PUs** | 157% | 162% | 200% | 165% |
| **5 PUs** | 196% | 218% | 250% | 200% |
| **6 PUs** | 234% | 260% | 300% | N/A |
| **7 PUs** | 274% | 305% | 350% | N/A |
| **8 PUs** | 320% | 354% | 400% | N/A |
| **9 PUs** | 368% | 400% | N/A | N/A |
| **10 PUs** | 400% | N/A | N/A | N/A |

We conducted four experiments with two parallel OpenMP applications: a 2D heat transfer problem [19] and a LU decomposition algorithm [20]. The heat transfer problem consists in

 ISBN: 978-1-61208-216-5 

solving a partial differential equation to determine the variation of the temperature within the heat conducting body. LU decomposition is a method to factorize a matrix as the product of a lower triangular matrix and an upper triangular matrix. This algorithm is a key step in several fundamental numerical algorithms in linear algebra such as solving a system of linear equations, inverting a matrix, or computing the determinant of a matrix.

The experiments were grouped in two sets, each set examining different issues. In Set A, three experiments test the ERHA efficiency in providing a homogeneous environment. In Set B, the impact of PMs load in application performance is evaluated. For each test, the applications were run 10 times and the results were combined to calculate mean, in a 95% confidence interval.

### A. ERHA Efficiency

In this section, three different test cases were used to validate the ERHA architecture and to analyze its efficiency using different configurations.

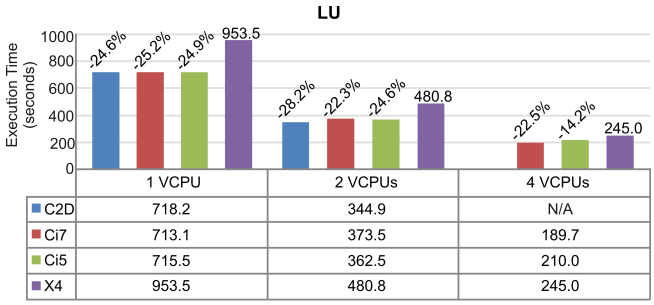
*1) Experiment 1:* In the first experiment we tested the architecture with the Limit Manager module disabled. The purpose of this experiment is to evaluate the difference in the PMs processing power. The LU decomposition and heat transfer applications were executed on a virtual environment allocated on one PM of each model (Ci5, Ci7, X4 and C2D). The tests were performed using VMs with 1, 2 and 4 VCPUs, except for C2D, which supports VMs with just 2 VCPUs.

Figure 4 shows the applications execution time for each configuration. It can be seen that both applications take longer in C2D for all configurations. The difference reaches 30.3% in the case of heat transfer application with 1 VCPU and 28,2% in the case of LU decomposition with 2 VCPUs. These results emphasize the need for a solution which considers the infrastructure's heterogeneity once they confirm the influence of the underlying PM on virtual machines performance and its applications.

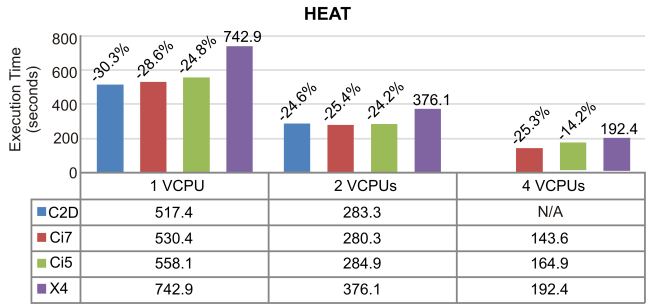
*2) Experiment 2:* In the second experiment, the Limit Manager features were evaluated. The LU decomposition and heat transfer applications were executed in VMs with 1, 2 and 4 VCPUs and setting the processing power to 1 PU per VCPU, totaling respectively 1, 2, and 4 PUs. The results are presented in Figure 5.

As expected, the execution time is greater than in Experiment 1, due the CPU limitation imposed by Limit Manager. Taking the PM X4 as basis, in LU decomposition the largest difference is 1.7% in processing time, while in the heat transfer problem the largest difference is 6%. The results show that performance variability is reduced if compared with Experiment 1.

*3) Experiment 3:* In the third experiment, the previous experiment we repeated using 2 PUs per VCPU. The VMs with 1, 2 and 4 VCPUs had processing power of respectively 2, 4 and 8 PUs. The results can be observed in Figure 6. The most relevant difference in execution time is 3.5% for LU

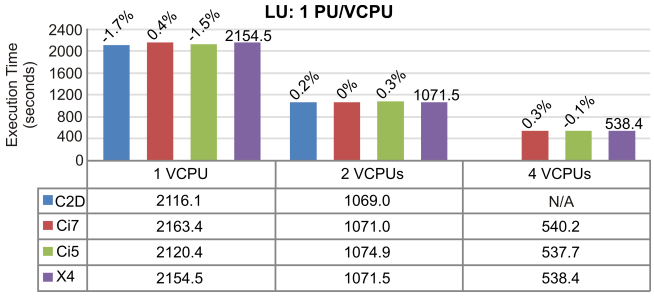



| | 1 VCPU | 2 VCPUs | 4 VCPUs |
|---|---|---|---|
| C2D | 718.2 | 344.9 | N/A |
| Ci7 | 713.1 | 373.5 | 189.7 |
| Ci5 | 715.5 | 362.5 | 210.0 |
| X4 | 953.5 | 480.8 | 245.0 |

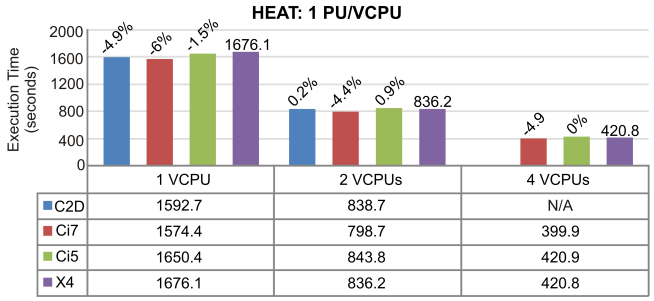



| | 1 VCPU | 2 VCPUs | 4 VCPUs |
|---|---|---|---|
| C2D | 517.4 | 283.3 | N/A |
| Ci7 | 530.4 | 280.3 | 143.6 |
| Ci5 | 558.1 | 284.9 | 164.9 |
| X4 | 742.9 | 376.1 | 192.4 |

Fig. 4. ERHA Efficiency - Execution time without CPU limiting




| | 1 VCPU | 2 VCPUs | 4 VCPUs |
|---|---|---|---|
| C2D | 2116.1 | 1069.0 | N/A |
| Ci7 | 2163.4 | 1071.0 | 540.2 |
| Ci5 | 2120.4 | 1074.9 | 537.7 |
| X4 | 2154.5 | 1071.5 | 538.4 |



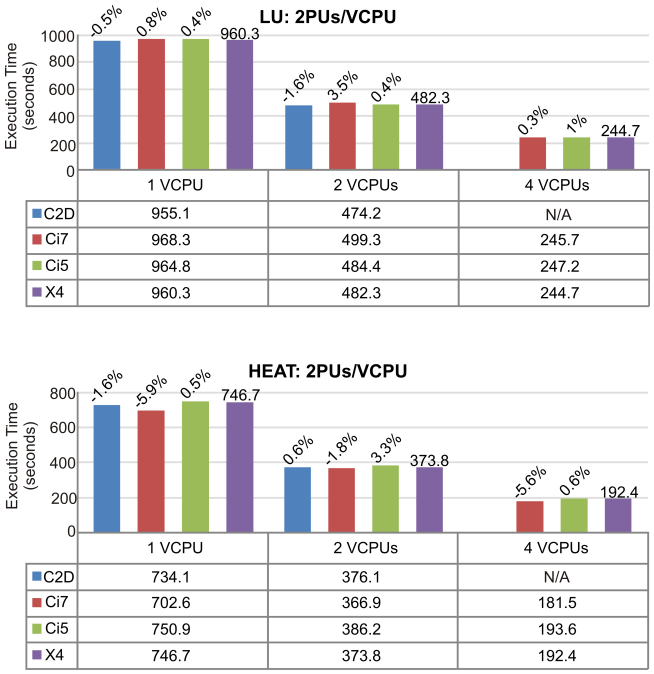

| | 1 VCPU | 2 VCPUs | 4 VCPUs |
|---|---|---|---|
| C2D | 1592.7 | 838.7 | N/A |
| Ci7 | 1574.4 | 798.7 | 399.9 |
| Ci5 | 1650.4 | 843.8 | 420.9 |
| X4 | 1676.1 | 836.2 | 420.8 |

Fig. 5. ERHA Efficiency - Execution time with CPU limiting enabled: 1 PU per VCPU

decomposition problem in case with 2 VCPUs and 5.9% for heat transfer problem in case with 1 VCPU.

The test confirmed the results of the second experiment, by proving the efficiency of the proposed solution to reduce the VMs performance variability commonly imposed by different PMs models. Furthermore, despite the execution time being larger when executing the applications with 4 VCPUs and 2 PUs per VCPU than in the case without CPU limitation, it is important to highlight that the more powerful processors still

 ISBN: 978-1-61208-216-5 




| | 1 VCPU | 2 VCPUs | 4 VCPUs |
|---|---|---|---|
| C2D | 955.1 | 474.2 | N/A |
| Ci7 | 968.3 | 499.3 | 245.7 |
| Ci5 | 964.8 | 484.4 | 247.2 |
| X4 | 960.3 | 482.3 | 244.7 |

| | 1 VCPU | 2 VCPUs | 4 VCPUs |
|---|---|---|---|
| C2D | 734.1 | 376.1 | N/A |
| Ci7 | 702.6 | 366.9 | 181.5 |
| Ci5 | 750.9 | 386.2 | 193.6 |
| X4 | 746.7 | 373.8 | 192.4 |

Fig. 6. ERHA Efficiency - Execution time with CPU limiting enabled: 2 PUs per VCPU

have resources enough to run more VMs (2 PUs free in Ci7), allowing other applications to run in that PM.

### B. Impact of Physical Machines Load

The objective of this experiment is to evaluate the efficiency of the architecture to provide performance isolation while running more than one VM per PM. In this experiment, two VMs were executed at the same time in each PM (Ci5, Ci7 and X4), one running the heat transfer application and the other running the LU reduction problem.

Two different configurations of VMs were used: VMs with 2 VCPUs and 4 PUs (2 PUs for each VCPU) and VMs with 4 VCPUs and 4 PUs (1 PU for each VCPU). Considering that two VMs will run on each PM, a total of 8 PUs will be used. The results can be observed in Figure 7, where the cases marked with *-extra load* represent the tests where 2 VMs are used in the same PM (competing for CPU resources). The results were compared with the previous experiments results (Figures 5 and 6), where a single VM per PM was used.

It can be observed that the largest differences between the execution time were 4,6% and 4,5% in the VMs with 2 and 4 VCPUs, respectively, both running the LU decomposition application in host Ci7. These results show that the solution provides a good performance isolation for 2 VMs running on the same PM. Further experiments must be performed to prove the ERHA's efficiency regardless the number of VMs per PM.

To sum up, the presented experiments were executed and configured easily with ERHA. The results demonstrate the solution's efficiency to deploy applications in clouds and to reduce the performance fluctuations despite being in a dynamic and heterogeneous environment.

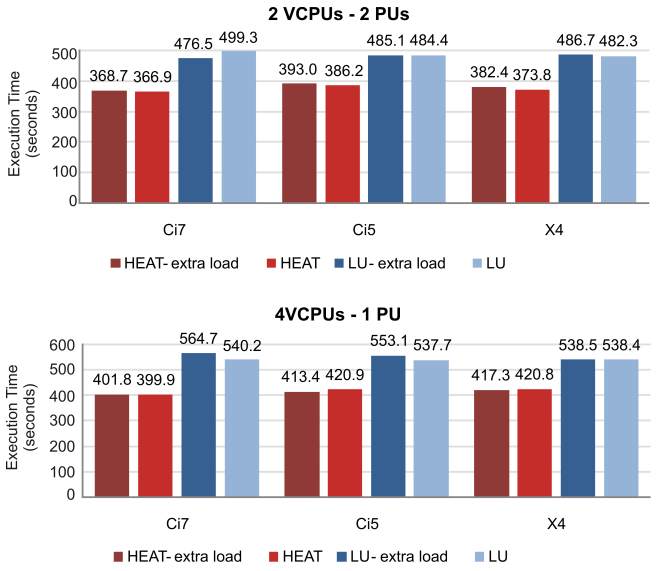



Fig. 7. Execution time using different loads in physical machines

## V. Conclusion and Future Work

This paper presented a solution to automate the deployment and execution of batch applications in clouds, providing mechanisms to create homogeneous virtual environments over private cloud middleware. The results described in Section IV confirmed the ERHA's efficiency in deploying applications in clouds and reducing the disparities in execution time using different PMs.

Although the presented tests have just used OpenMP applications, ERHA allows to run sequential, shared-memory and distributed memory parallel applications. For example, it is possible to run MPI applications in a homogeneous virtual cluster with all credentials for SSH communications, automatically created and managed by the architecture.

The architecture is useful in the cases where the researchers expect comparable performance for their applications, independent of the physical resources used. This feature is quite important for repeatability of experiments and results. Furthermore, considering the uniform processing power allocations provided by ERHA, it is possible to reduce the performance variability in VMs' migrations between different physical machine types.

To conclude, the main contribution of this paper was the reduction of the impact of the data center heterogeneity in the VMs performance. In addition, we proposed mechanisms to enable running applications in clouds in an easy and uniform way, abstracting the infrastructure and middleware complexities.

The next step in our research is to extend the architecture implementation to work with other cloud middleware and new application types such as MapReduce. We also intend to create new scheduling policies and implement an interface to permit users to dynamically change the amount of PUs and VCPUs allocated to VMs.

 ISBN: 978-1-61208-216-5 

## ACKNOWLEDGMENTS

This work is supported by FUNCAP, CAPES and INCT-MACC (CNPq process 573710/2008-2).

## REFERENCES

[1] G. V. Mc Evoy, B. Schulze, and E. L. M. Garcia, "Performance and deployment evaluation of a parallel application on a private cloud," *Conc. and Comp.: Prac. and Exp.*, vol. 23, pp. 2048–2062, Dec. 2011.

[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." *October*, vol. 2, no. 2.40, pp. 2–34, 2008.

[3] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience," in *Proc. of the 2008 10th IEEE Intl Conf. on High Perf. Comp. and Comm.*, ser. HPCC '08. IEEE Computer Society, 2008, pp. 825–830.

[4] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman, "Experiences using cloud computing for a scientific workflow application," in *Proc.of the 2nd Intl Workshop on Scientific cloud computing*, ser. ScienceCloud '11. ACM, 2011, pp. 15–24.

[5] Y. Simmhan, C. Van Ingen, G. Subramanian, and J. Li, "Bridging the gap between desktop and the cloud for escience applications," *IEEE 3rd Intl Conf. on Cloud Computing*, pp. 474–481, 2010.

[6] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proc. VLDB Endow.*, vol. 3, pp. 460–471, September 2010.

[7] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *IEEE/ACM 11th Intl Symposium on Cluster, Cloud and Grid Computing*, may 2011, pp. 104–113.

[8] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud." in *Proc. of IEEE Intl Conf. on Cloud Computing Technology and Science*, 2010, pp. 159–168.

[9] Y. El-Khamra, H. Kim, S. Jha, and M. Parashar, "Exploring the performance fluctuations of hpc workloads on clouds," in *Proc. IEEE Second Intl Conf. on Cloud Computing Technology and Science*, 30 2010-dec. 3 2010, pp. 383–387.

[10] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Proc. of the ACM/IFIP/USENIX 2006 Intl Conf. on Middleware*, ser. Middleware '06. Springer-Verlag New York, Inc., 2006, pp. 342–362.

[11] W. Dawoud, I. Takouna, and C. Meinel, "Elastic vm for cloud resources provisioning optimization," in *Advances in Comp. and Comm.*, ser. Comm. in Comp. and Information Science, A. Abraham, J. Lloret Mauri, J. F. Buford, J. Suzuki, and S. M. Thampi, Eds. Springer Berlin Heidelberg, 2011, vol. 190, pp. 431–445.

[12] A. Sangpetch, A. Turner, and H. Kim, "How to tame your vms: an automated control system for virtualized services," in *Proce. of the 24th Intl Conf. on Large installation system administration*, ser. LISA'10. USENIX Association, 2010, pp. 1–16.

[13] C. Bunch, N. Chohan, C. Krintz, and K. Shams, "Neptune: a domain specific language for deploying hpc software on cloud platforms," in *Proc. of the 2nd Intl workshop on Scientific cloud computing*, ser. ScienceCloud '11. ACM, 2011, pp. 59–68.

[14] J. Mirkovic, T. Faber., P. Hsieh, G. Malayandisamu, and R. Malavia, "Dadl: Distributed application description language," USC/ISI, Tech. Rep. ISI-TR-664, 2010.

[15] Nimbus Project, http://www.nimbusproject.org/, [retrieved: june, 2012].

[16] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, "Quantitative comparison of Xen and KVM," in *Xen summit*. Berkeley, CA, USA: USENIX association, Jun. 2008.

[17] P. A. L. Rego, E. F. Coutinho, D. G. Gomes, and J. N. de Souza, "Faircpu: Architecture for allocation of virtual machines using processing features," in *Proc. of Fourth IEEE Intl Conf. on Utility and Cloud Computing*, Dec 2011, pp. 371 –376.

[18] ERHA Project, http://www.inf.ufpr.br/ggalante/erha, [retrieved: june, 2012].

[19] J. H. Lienhard and J. H. Lienhard, *A Heat Transfer Textbook - 3rd ed.* Phlogiston Press: Cambridge, Massachusetts, 2008.

[20] D. Poole, *Linear Algebra: A Modern Introduction*. Cengage Learning, 2006.

 ISBN: 978-1-61208-216-5 