On the Elasticity of Parallel Components in a Cloud of High Performance Computing Services

João Marcelo Uchôa de Alencar[1] **and Francisco Heron de Carvalho Junior**[2]

[1] Campus de Quixadá
Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003 – Quixadá – CE – Brazil

[2]Pós-Graduação em Ciência da Computação (MDCC)
Universidade Federal do Ceará (UFC)
Campus Universitário do Pici, Bloco 912 – Fortaleza – CE – Brazil

{joao.marcelo,heron}@lia.ufc.br

Abstract. *Cloud computing offers virtually unlimited set of resources and flexibility to allocate them through elasticity. But cloud limitations, such as the complexity of configuration and environment dynamicity, may jeopardizes the assurance of QoS requirements. HPC Shelf is a cloud of HPC services that employs a component-oriented architecture to describe hardware and software resources of parallel computing systems. We design a framework for HPC Shelf that employ cloud elasticity concepts for keeping the values of QoS metrics of parallel computing systems inside an acceptable range, enabling adaptations to fulfill the QoS contract restrictions. In our evaluation, using a linear algebra application, we show how HPC Shelf takes advantage of cloud elasticity to reinforce QoS requirements, rectifying assumptions from ill-defined QoS models.*

1. Introduction

Complexity and heterogeneity are recurrent words in debates about today's high-performance computing (HPC) landscape. Clusters, grids, and clouds have increased the possibilities of parallel computing infrastructures. Developers must write code that exploit particularities of different types of systems, heterogeneous with respect to processors, accelerators, memory hierarchies, and interconnections. In fact, they need fine control over how the software uses hardware resources to achieve peak performance.

HPC Shelf [1] is an ongoing project that aims at reducing the complexity of managing both hardware and software artifacts of heterogeneous HPC environments. It employs component technology to represent these artifacts and encapsulates functional and non-functional concerns by using the abstraction of a contextual contracts.

HPC Shelf has been extended with the specification of QoS (quality of service) requirements of components through contextual contracts. A system component represents a pair of components: a computation and a virtual platform where the computation is placed. The QoS requirements feed a ranking algorithm that classifies system components according to their ability to fulfill the QoS requirements. HPC Shelf defines contracts using estimation functions provided by component developers and platform maintainers.

The main contribution reported in this paper is a framework that uses elasticity to steer the execution towards contract fulfillment in the event of ill-defined requirements and unpredictable behavior from cloud environments. It allows embedding the expression of different scaling techniques to support a broad range of HPC applications.

The rest of the paper comprises other five sections. Section 2 discusses related works. Section 3 details HPC Shelf and the use of contextual contracts augmented with QoS requirements. Section 4 describes the framework for reconfiguration of HPC applications. Section 5 presents an experimental study to partially validate the artifact proposed in this paper. Section 6 presents our conclusions and point to future works.

2. Related Works

McInnes *et al* introduce Computational Quality of Service (CQoS) with method adaptation [2] for CCA (*Common Component Architecture*) frameworks, where analysis infrastructure continuously monitors the executing computation. If the component does not provide the required performance, it is replaced by another one with the same functionality, but keeping unchanged the parallel computing infrastructure where it executes.

For GCM (*Grid Component Model*) [3], *behavioral skeletons* deserves attention. They allow developers to define a hierarchy of components where each component has an internal autonomic agent for reconfiguration of processes and threads. The user provides requirements to the top-level element of the composition. Through filtering and delegation, the distributed system changes its parallelism level for ensuring the QoS. As CQoS, the focus is on adapting the software component, not the platform.

For elasticity in HPC, the basic alternative is dynamic resource management [4, 5, 6], where platform managers create virtual clusters in an IaaS cloud. The users submit tasks or jobs in the same way they do for traditional supercomputers. However, instead of a fixed set of processing nodes, the manager analyzes the submission queue and creates only enough virtual nodes to satisfy the user requirements. A virtual cluster may expand and shrink according to the current workload, not requiring to reconfigure tasks or jobs.

Other works allow developers to change the resources of the parallel computing platform during execution [7, 8, 9]. For that, a runtime environment supports an API for controlling the creation, removal, and reconfiguration of virtual nodes. Each program may employ its own technique, and there is no native model for code reuse. Some works do not require developer intervention, providing a single procedure for load balancing [10].

The existing solutions over a component-based architecture are restricted to software adaptation. When the focus shifts to resource adaptation, there is no standard programming model, and different applications may require different techniques. Since HPC Shelf aims to support applications from several domains, it is necessary to provide ways of reusing existing concepts and personalizing techniques for emerging scenarios.

3. HPC Shelf: Towards Cloud-Based HPC Services

HPC Shelf [1] is a component-oriented platform for deploying cloud computing applications that demand large-scale parallel computing infrastructures comprising a set of clusters for executing solutions to computationally challenging problems specified by domain specialists. HPC Shelf complies to the Hash component model [11]. For our purposes in this paper, among the component kinds it supports, we are only interested in **virtual platforms**, which represent distributed-memory parallel computers, and **computations**, which implement parallel algorithms tuned for a class of virtual platforms.

Stakeholders In HPC Shelf, stakeholders with complementary interests act together to provide HPC services to domain **specialists**, through applications. Applications are supplied by **providers**, providing domain-specific high-level interfaces for problem specification, hiding parallel computing systems built of components to solve them. Computational components are developed by **developers**. They tend to be application-independent in order to attend different providers. Finally, virtual platform components are supplied by infrastructure **maintainers**, over computing infrastructures such as IaaS providers.

Architecture HPC Shelf comprises instances of three architecural elements: *Front-End*, *Core* and *Back-End*. The *Front-End* is SAFe (*Shelf Application Framework*), from which applications may be derived [1]. It is a SWfMS (*Scientific Workflow Management System*) where worflows are described using SAFeSWL (*SAFe Scientific Workflow Language*). The *Core* controls the life-cycle of components, as well as supporting the system of *contextual contracts* described in the next section. Finally, a *Back-End* is a service that manages a certain parallel computing infrastructure (e.g. IaaS clouds, on-premise clusters, etc), over which it is responsible to instantiate and monitor virtual platforms.

3.1. Contractual QoS and Resolution Classification

The resolution of contextual contracts makes it possible the selection of components that fullfill contractual requirements [12]. But which is the better choice in the list of selected components ? This question has been responded by introducing *quality criteria* for classifying them, by adding QoS parameters to contextual contracts of HPC Shelf.

Norris *et al* [13] argue that CBHPC environments should deal with the ***performance*** and ***accuracy*** of components for ensuring computational QoS (CQoS). Indeed, *performance* is the execution time of a component for a given input, while *accuracy* asserts how close the solution obtained by a component is from the expected results. We include **efficiency**, denoting the level of resource utilization, as another desired goal, such as the load average of processing nodes in a cluster. Moreover, we include **monetary cost** and **power consumption**, now relevant in the context of clouds. The former is the amount of money that specialists must pay to maintainers for the resources they allocate during execution, while the later measures, in kWh, the power consumption of these resources.

Developers are responsible to setup QoS parameters of computation components in their contextual contracts since they know the programming language, algorithms and data structures employed. For that, they may use *estimation functions* to calculate arguments for each QoS parameter, from the arguments applied to the contextual parameters that describe application requirements and platform features. However, the specification of estimation functions is a difficult task. For performance requirements, analytical models are a way of studying the behavior of computations through equations that quantify the amount of work done for each processing unit, memory access pattern, communication load, and other metrics [14, 15]. Other approaches employ machine learning techniques [16]. HPC Shelf only expects that estimation functions return arguments to QoS contextual parameters of a given computational system contract (CSC), comprising contracts for both the virtual platform (PC) and the computation components (CC). For instance, the array $QoS = [a, e, t, c, p]$ has values for accuracy, efficiency, execution time, cost and power consumption, respectively, calculated through estimation functions.

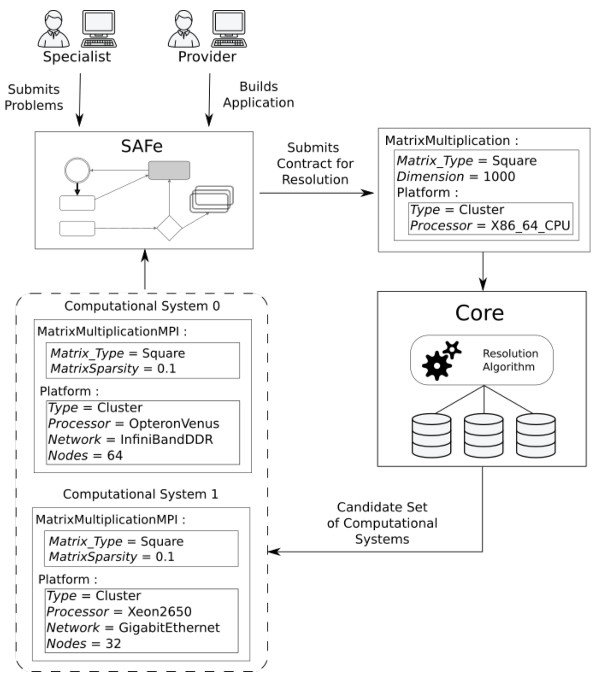

Figure 1. Defining a list of candidate Computational Systems.

The developer is free to choose among analytical models, simulations, machine learning techniques, etc. The estimate has to provide a guideline for adapting the execution of parallel computing systems to meet QoS requirements. If the application makes a feasible choice of initial computational system, its adaptation to reach the QoS requirements is easier. That is the reason why a way of ranking components is necessary.

We define a particular approach for classifying candidate components that we will use throughout this paper. Let $CSC_i[PC_i, CC_i]^{\ i=1...n}$ be the set of candidates returned by the selection stage of resolution. Their estimation functions, specified by component developers, are the arrays $QoS_i^{\ i=1...n}$, respectively, further divided in two:

$$QoS_i^{max} = [a_i, e_i] \text{ and } QoS_i^{min} = [t_i, c_i, p_i]$$

QoS_i^{max} has a value for each QoS context parameter with *additive subtyping* (the higher value, the better), whereas QoS_i^{min} has a value for that ones with *subtractive subtyping* (the lower value, the better). Let $max(x)$ and $min(y)$, for $x \in \{a, e\}$ and $y \in \{t, c, p\}$, the maximum and minimum value for each QoS context parameter, either direct or inverse. We can normalize the array valuations by:

$$QoS_i^{max} = [\frac{a_i - min(a)}{max(a) - min(a)}, \frac{e_i - min(e)}{max(e) - min(e)}]$$
$$QoS_i^{min} = [\frac{t_i - min(t)}{max(t) - min(t)}, \frac{c_i - min(c)}{max(c) - min(c)}, \frac{p_i - min(p)}{max(p) - min(p)}]$$

Application providers may inform two arrays for each parallel computing system, setting up the weight of each QoS parameter in classification, in such a way they may customize the classification policy according to their requirements.

$$W_{max} = [w_a, w_e], w_a + w_e = 1 \text{ and } W_{min} = [w_t, w_c, w_p], w_t + w_c + w_p = 1$$

So, the ranking value for each system $CSC_i[PC_i, CC_i]$ is:

$$R_i = (w_a * a_i + w_e * e_i) + (1 - (w_t * t_i + w_c * c_i + w_p * p_i))$$

The candidates are sorted according to $R_i\ ^{i=1 \ldots n}$, so that the application choose the first candidate in the ranking. As an example of the proposed ranking method, consider three candidate components with the following QoS contracts:

$$QoS_0^{min} = [0.7, 0.8], QoS_0^{max} = [1600, 10, 3]; QoS_1^{min} = [0.9, 0.9], QoS_1^{max} = [1800, 6, 4];$$
$$QoS_2^{min} = [0.9, 0.7], QoS_2^{max} = [1500, 15, 5]$$

By normalizing the array valuations, we have:

$$QoS_{0max} = [0, 0.5], QoS_{0min} = [0.34, 0.45, 0]; QoS_{1max} = [1, 1], QoS_{1min} = [1, 0, 0.5];$$
$$QoS_{2max} = [1, 0], QoS_{2min} = [0, 1, 1]$$

If an application applies $W = [0.7, 0.3, 0.8, 0.1, 0.1]$, prioritizing accuracy and execution time, $R_0 = 0.84$, $R_1 = 1.15$, $R_2 = 1.50$. The candidate with highest ranking value (CSC_2) has the best accuracy (0.9) and lowest estimated execution time (1500s).

4. A Framework for Elastic Reconfiguration in HPC Shelf

When the instantiation of a computation component is triggered in a parallel computing system, its contextual contract must have been previously resolved, as well as its host virtual platform, since virtual platforms must be instantiated before the computations they host. SAFe instantiates components through the Core's services. In the case of virtual platforms, the Core invokes the *Backend* service of its maintainer. In turn, for instantiating computations, the Core invokes the services of their host virtual platforms.

For application providers, the execution of a component must comply with the QoS requirements of its contextual contract. However, the dynamicity of modern infrastructures may impose restrictions for this. For instance, for a virtual platform over a IaaS cloud, interference may arise due to multitenancy [17]. Despite the nodes of the virtual platform may be started with exclusive use of a host machine, another virtual machine of a different application may be allocated to the same machine further. The sharing of the memory bus, network, and caches may constitute bottlenecks that may jeopardize QoS parameters. Although commercial IaaS clouds now offer dedicated servers aimed at HPC [18], many users do not have money to pay for them. Also, many institutional built private clouds with open source solutions that are not optimized for HPC [19].

Imprecise estimation functions provided by developers and maintainers may also jeopardize the fulfillment of QoS requirements, since desired QoS values may not be achieved through the initial set of resources allocated. In this paper, we are interested

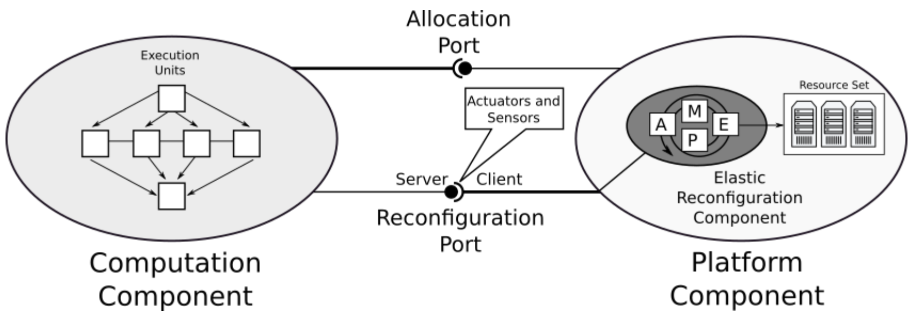

Figure 2. Elastic System Component Architecture

in designing an automated adjustment mechanism to fulfill QoS requirements, where the original contracts of components guide reconfiguration. For that, we introduce a notion of elasticity to component-oriented parallel computing, meaning the ability to reconfigure virtual platforms and computations by adding or removing resources they employ.

For elasticity, we introduce *elastic contextual contracts*, in opposite to *rigid* ones, supporting numerical range qualifiers. They make it possible to specify a range of acceptable arguments for a numerically valued parameter of a contextual signature. For example, consider an illustrative contract for a virtual platform supported by a maintainer:

$$\textsc{Cluster}\begin{bmatrix} \textbf{\textit{number\_of\_nodes}} = [24\text{-}48], \\ \textbf{\textit{node}} = \textsc{Xeon}[\textbf{\textit{model\_type}} = \textsc{Xeon}2650, \textbf{\textit{cores\_per\_node}} = [2\text{-}8], \textbf{\textit{memory\_size}} = [16\text{-}32]] \end{bmatrix}$$

With a range qualifier typing the contextual parameter ***number_of_nodes***, instead of fixing the number of nodes of a virtual platform, the maintainer provides a range $[24, 48]$, specifying minimum and a maximum limits. The *Core* may instantiate the virtual platform with any number of nodes in the range and vary it during the execution, between 24 and 48, in order to fulfill QoS requirements. This is an example of *horizontal elasticity*. For *vertical elasticity*, the maintainer also supply ranges for ***memory_size*** and ***cores_per_node***, varying the amount of memory and number of cores in each node.

4.1. Elastic System Components

Feitelson and Rudolph have proposed a taxonomy for parallel tasks, where they can be classified in *ridig*, *moldable*, *malleable* and *evolutive* [20]. HPC Shelf already supports both ridig and moldable computation components, so that they may be statically adapted to the resources of the target (non-elastic) virtual platform. This is the usual approach in parallel programming for cluster computing. In this paper, we define a notion of *elastic system component* through a pair of a *malleable computation component* and an *evolutive virtual platform*. An elastic system component allows dynamic adjustments in the set of resources employed by virtual platforms. For example, changing its number of nodes and, consequently, the number of units of hosted computation components. The evolutive virtual platform decides when to apply a reconfiguration from the state of hosted computations and the underlying infrastructure, expecting that the malleable computation component reacts by adjusting itself to the new resource set, when necessary.

Figure 2 depicts the architecture of an elastic system component. When the *Core* instantiates a computation, it retrieves the initial set of resources from the allocation port of the virtual platform, by looking at elastic contextual arguments. Such a resource set may describe, for example, the number of available nodes along with their hostnames, the number of cores per node, and the amount of memory per node, and so on. The resolution algorithm ensures that the computation component is compatible with the available resource set. The virtual platform starts an inner reconfiguration component that runs the four phases of the MAPE loop [21]: monitoring, analysis, planning, and execution.

The computation component provides two ports to the virtual platform: *monitoring* and *reconfiguration*. From the monitoring port, the virtual platform may query sensor variables continuously updated by the computation for assisting reconfiguration decisions. From the reconfiguration port, the platform may trigger adaptations in the computation.

There is a sensor variable for returning a measure of the current solution accuracy. It is heavily reliant on the algorithm implemented by the computation. It may have no meaning, like in exact numerical algorithms. However, for approximation algorithms, the developer may create mechanisms to update this variable. We define it has a value in the interval $[0.0, 1.0]$, whose update is decided by the developer. If the actual accuracy is below the threshold specified in the QoS contract, the virtual platform may provide more resources to the computation (scale-up elasticity). In turn, if the accuracy is above the threshold, the virtual platform may reclaim resources from the computation (scale down elasticity) in order to minimize costs, energy consumption, among other reasons.

Another sensor variable may measure *progress*, updated at each step the computation completes. For example, if the computation must process N data items, it is updated to the number of items it has processes over N at each processed item. As the accuracy sensor, the progress is in the range $[0.0, ..., 1.0]$ whose update is decided by the developer.

4.2. Reconfiguration Phases

During the analysis phase, the MAPE loop estimates execution time from the value of the progress variable and the known startup time of the computation. If the execution time is above the contractual threshold, it may be necessary to scale up resources. However, in certain circumstances, due to the scalability characteristics of the parallel algorithm being executed, scaling down may be the best choice. The predicted termination time and the current resource set allow measuring estimated cost and power consumption, which HPC Shelf may also use for reconfiguration decisions according to cost contextual arguments.

Along with the computation state, the MAPE loop retrieves the resource state. For that, the virtual platform queries each node for its average load in the last minute. The efficiency of the resource set is the sum of the load average from each machine over the total number of cores. If the efficiency is below the contractual threshold, nodes may be released to increase utilization. If it is above, nodes may be added to decrease utilization.

The reconfiguration procedure taken to fulfill the requirements of a given QoS parameter may affect the fulfillment of others. For example, for embarrassingly parallel computations, the execution time may be reduced by employing more resources, thus increasing costs. The first step to deal with conflicting goals in the planning phase is to sort out values that are within an acceptable range. For that, QoS contracts have an α value in $[0, 1]$. If an argument to a QoS parameter is inside $[(1 - \alpha) * QoS[parameter], (1 + \alpha) * QoS[parameter]]$, then it cannot guide the reconfiguration. If all the parameters have arguments within the acceptable range, the loop proceeds. Otherwise, in the second step, HPC Shelf decides which parameter to choose according to the weight arrays given in

Section 3.1. The virtual platform computes the difference between the predicted argument for each parameter and its requirement of the QoS contract and divides it by the contract value. Then, its absolute value is multiplied by the weight value of this parameter. The parameter with the highest value will guide the reconfiguration. This approach does not eliminate conflicts but ensures prioritization of QoS requirements of the provider.

In the planning phase, the reconfiguration exhausts the possibility of vertical elasticity, before it proceeds to horizontal elasticity. Resources are added or removed in units. For example, cores are added or transferred to the nodes in the resource set at the rate of one core per reconfiguration. If the decision procedure reaches the limits of the contract, then horizontal elasticity adds or removes nodes, also at the rate of one per reconfiguration. If it reaches the boundaries of the agreement again, HPC Shelf emits a warning to the provider, but the execution will proceed unless the application decides to cancel it.

For the execution phase, the virtual platform sends a new resource set to the computation component through an *actuator method* when it decides to perform a reconfiguration. This new resource set may differ from the previous one regarding the number of cores per node (vertical elasticity) or node count (horizontal elasticity). Developers willing to support malleable reconfiguration of their computations must create components that support reconfiguration through the actuator method.

5. Experimental Evaluation

As a small-scale case study, we demonstrate how the elasticity framework propose in this paper may horizontally reconfigure an elastic system component.

5.1. Environment Setup

The computation component we have developed has a user service port through which it takes a list of square matrix pairs. Then, it multiplies the matrices of each pair, and returns a list of files containing the resulting matrices through another operation of the same port. The order of each matrix pair may vary. For example, the input list

$$\left[(A_0, B_0)_{1000\times1000}, (A_1, B_1)_{2000\times2000}, (A_2, B_2)_{1500\times1500}\right]$$

causes three multiplications of matrices of orders 1000, 2000 e 1500, respectively. So, the output is $[File_0, File_1, File_2]$, a list of files with matrices of the same orders.

The computation component may return the progress of the execution. Internally, it is an MPI program. In each multiplication $A \times B$, the root unit scatters the lines of A and the corresponding columns of B across the worker units. Then, each worker performs a partial multiplication. Finally, the root gathers the results in the output file.

We have employed the Python language with OpenMPI 1.10.2 (mpi4py) and the numpy package for matrix calculation. Also, we have used an OpenStack-based cluster as a parallel computing platform, with six physical servers as compute nodes. Each node has two Intel Xeon CPU E5-2650 v3 2.30GHz, with a total os 20 cores and 64GB of memory. The network that interconnects nodes was Gigabit Ethernet. The OpenStack version was Liberty, with KVM as virtualization technology.

Since KVM does not support vertical elasticity, only horizontal elasticity is considered. In order to retain granularity control, we have used a virtual machine with two cores and 4GB of RAM, running the Ubuntu Server, version 16.04.02 LTS.

Table 1. Computational Component Behavior without Reconfiguration.

VMs	Matrix Size	Elapse	Load	Cost
2	5000	1131.11	1.75	2264.22
2	10000	7565.08	1.93	15130.17

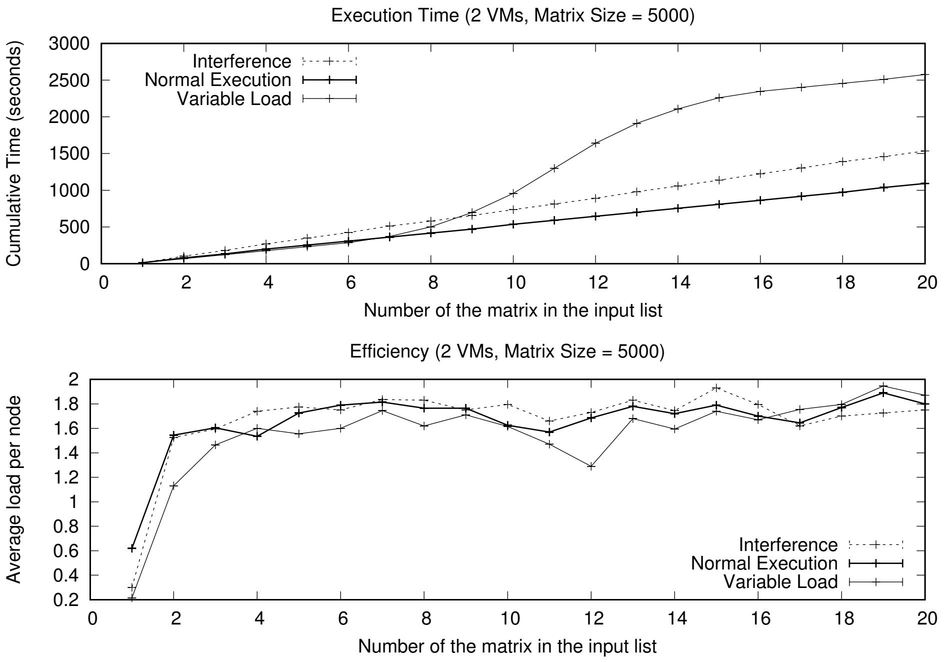

Figure 3. System Behavior without Reconfiguration.

5.2. System Behavior without Reconfiguration

We have submitted lists of twenty dense floating point matrices for multiplication, varying the number of nodes of the virtual platform and the order of the matrices. The system deploys the virtual machines in a round-robin manner among the servers. Table 1 shows the results for elapsed time, average load (efficiency) per node, and total cost. We do not take accuracy into account, since the algorithm has fixed precision and power consumption. However, as explained in Section 4.1, power consumption and cost are proportional to the number of nodes, so that we may infer the energy requirements from the price value.

Figure 3 depicts an execution with two VMs and twenty dense matrix multiplications with an order of 5000. The typical execution scenario is the same we have used to provide the data in Table 1, with one VM per server. We have made such a change for creating interference in one of the physical servers to simulate the effects of multi-tenancy. During the execution of the same workload, we have created a platform with ten VMs restricted to one of the servers, colocated to an existing VM from the original platform, overloading the server, with an impact on the execution of the computational component. The interference increases the execution time from 1132s to 1675s.

We have also simulated ill-defined QoS contracts by creating an input list with matrices of different orders. Such a scenario employs a list where the first and last five matrix pairs have the order 5000, but from the sixth to the tenth pair, the size grows from 6000

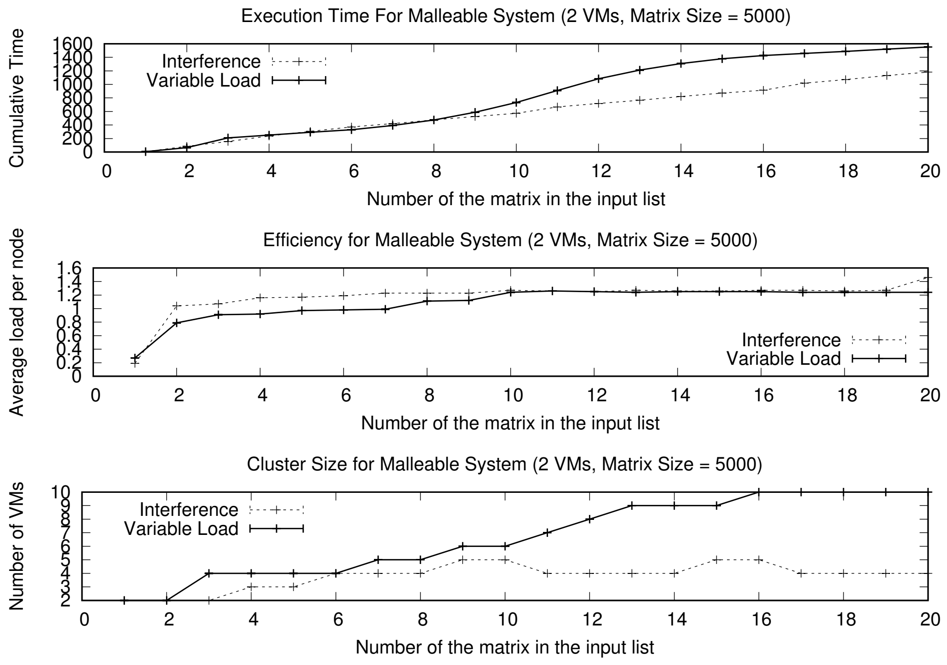

Figure 4. System Behavior with Reconfiguration and Execution Time Priority.

to 10000, decreasing after that by 1000 until reaching 6000 again. Figure 3 shows that this Variable Load scenario is more costly than the Interference one, reaching 2670s, because multiplying matrices of order 10000 with 2 VMs causes memory paging.

5.3. System Behavior with Reconfiguration

In order to demonstrate how HPC Shelf may reconfigure virtual platforms and computation components, we define a QoS contract that requires a execution time of 1132s, an efficiency of 1.75 and a cost (in a hypothetical monetary unit) of \$2264, with priority weights of 0.8, 0.1 and 0.1, respectively. The reconfiguration system was configured with a monitoring interval of 10s, a reconfiguration interval of 60s, and an α of 0.1.

The expected values come from initial results of the execution without reconfiguration for 2 VMs and twenty matrices of order 5000, giving much higher priority to execution time. It is noteworthy that the system does not make assumptions about the scenario (Interference or Variable Load). It is only concerned with contract fulfillment.

Figure 4 shows that while not reaching the exact values required by the contract, the reconfiguration algorithm steers the system towards them. Without reconfiguration, the execution time for the interference scenario is 1675s. With reconfiguration, the computation finishes around 1288s, closer to the 1132s imposed by the contract. The same happens for the variable load scenario since more virtual machines available implies in more free memory. In such a case, the time decreases from 2670s to 1610s.

The downside of reconfiguration is at the bottom of Figure 4. For ensure a proper execution time, the reconfiguration system allocates new virtual machines, increasing the cost. The Interference scenario reaches an expense of \$5762 while the Variable Load one reaches a peak of \$15346. Unfortunately, these values are much higher than the contract value of \$2264, due to the priority given to execution time.

6. Conclusion and Future Works

In this paper, we propose a reconfiguration system aimed at ensuring that QoS requirements of applications, specified in contracts, are satisfied during execution of parallel computing systems they create, deploy and execute. It is implemented on top of HPC Shelf, a platform of HPC services in cloud. Instead of just swapping software components, like other related systems, our system delegates reconfiguration decisions to virtual platforms where they are placed, also viewed as components, in order to form an elastic notion of system component, comprising an evolutive virtual platform and a malleable computation component.

Unlike other elasticity solutions for HPC, the architecture of our proposal is highly concerned with separation of concerns, so that the stakeholders around an HPC Shelf-based environment (specialists, providers, developers and maintainers) know precisely their roles in reconfiguration support. It is also noteworthy that the proposed system may be adapted to other component-based platforms, since it does not make strong assumptions about HPC Shelf abstractions on top of which it has been firstly conceived.

For future work, we want to expand the evaluation with broader and larger scale scenarios, considering different sets of priorities. We also want to explore more realistic case studies, such as components that implement approximation methods, where accuracy varies by resource set. Finally, we intend to support evolutive computation components, where the computation is also involved in reconfiguration decisions.

References

[1] F. H. de Carvalho Junior, J. C. Silva, and A. B. O. Dantas, "A Scientific Workflow Management System for Orchestration of Parallel Components in a Cloud of Large-Scale Parallel Processing Services," *Science of Computer Programming*, vol. 173, pp. 95–127, Mar. 2019.

[2] L. C. McInnes, J. Ray, R. Armstrong, T. L. Dahlgren, A. Malony, B. Norris, S. Shende, J. P. Kenny, and J. Steensland, "Computational quality of service for scientific cca applications: Composition, substitution," and reconfiguration. Technical Report ANL/MCS-P1326-0206, Argonne National Laboratory, Tech. Rep., 2006.

[3] F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, and C. Pérez, "Gcm: a grid extension to fractal for autonomous distributed components," *Annals of Telecommunications-annales des télécommunications*, vol. 64, no. 1-2, pp. 5–24, 2009.

[4] A. Gupta, "Techniques for Efficient High Performance Computing in the Cloud," 2014. [Online]. Available: http://hdl.handle.net/2142/50718

[5] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid Computing—Where HPC meets grid and Cloud Computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 440–453, May 2011. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.11.003

[6] M. Caballer, C. de Alfonso, F. Alvarruiz, and G. Moltó, "EC3: Elastic Cloud Computing Cluster," *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1341–1351, Dec. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000013001141

[7] G. Mencagli, M. Vanneschi, and E. Vespa, "Control-theoretic adaptation strategies for autonomic reconfigurable parallel applications on cloud environments," in *High Performance Computing and Simulation (HPCS), 2013 International Conference on*, July 2013, pp. 11–18.

[8] A. Raveendran, T. Bicer, and G. Agrawal, "A Framework for Elastic Execution of Existing MPI Programs," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, May 2011, pp. 940–947.

[9] R. R. Righi, V. F. Rodrigues, C. A. da Costa, G. Galante, L. C. E. de Bona, and T. Ferreto, "AutoElastic: Automatic Resource Elasticity for High Performance Applications in the Cloud," *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 6–19, Jan 2016.

[10] R. da Rosa Righi, V. F. Rodrigues, G. Rostirolla, C. A. da Costa, E. Roloff, and P. O. A. Navaux, "A lightweight plug-and-play elasticity service for self-organizing resource provisioning on parallel applications," *Future Generation Computer Systems*, pp. –, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17302339

[11] F. H. de Carvalho Junior and R. D. Lins, "Separation of Concerns for Improving Practice of Parallel Programming," *INFORMATION, An International Journal*, vol. 8, no. 5, pp. 621–638, Sep. 2005.

[12] F. H. de Carvalho Junior, C. A. Rezende, J. C. Silva, W. G. Al Alam, and J. M. U. de Alencar, "Contextual Abstraction in a Type System for Component-Based High Performance Computing Platforms," *Science of Computer Programming*, vol. 132, pp. 96–128, 2016.

[13] B. Norris, J. Ray, R. Armstrong, L. C. McInnes, D. E. Bernholdt, W. R. Elwasif, A. D. Malony, and S. Shende, *Computational Quality of Service for Scientific Components*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 264–271. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24774-6_23

[14] G. Marin and J. Mellor Crummey, "Cross-architecture Performance Predictions for Scientific Applications Using Parameterized Models," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '04/Performance '04. New York, NY, USA: ACM, 2004, pp. 2–13. [Online]. Available: http://doi.acm.org/10.1145/1005686.1005691

[15] S. Lee, J. S. Meredith, and J. S. Vetter, "COMPASS: A Framework for Automated Performance Modeling and Prediction," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS. New York, NY, USA: ACM, 2015, pp. 405–414.

[16] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An Approach to Performance Prediction for Parallel Applications," in *Proceedings of the 11th International Euro-Par Conference on Parallel Processing*, ser. Euro-Par'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 196–205. [Online]. Available: http://dx.doi.org/10.1007/11549468_24

[17] S. Salaria, K. Brown, H. Jitsumoto, and S. Matsuoka, "Evaluation of HPC-Big Data Applications Using Cloud Platforms," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGrid'17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 1053–1061. [Online]. Available: https://doi.org/10.1109/CCGRID.2017.143

[18] M. Gilani, C. Inibhunu, and Q. H. Mahmoud, "Application and Network Performance of Amazon Elastic Compute Cloud Instances," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, Oct. 2015, pp. 315–318.

[19] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: Openstack and opennebula," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*. IEEE, 2012, pp. 2457–2461.

[20] D. G. Feitelson and L. Rudolph, "Toward Convergence in Job Schedulers for Parallel Supercomputers," in *Lecture Notes In Computer Science, vol. 1162 (Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'96)*, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 1–26.

[21] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.