Dimensioning the Virtual Cluster for Parallel Scientific Workflows in Clouds

Daniel de Oliveira[1], Vitor Viana[3]
[1]IC/UFF
danielcmo@ic.uff.br

Eduardo Ogasawara[2]
[2]CEFET/RJ
eogasawara@cefet-rj.br

Kary Ocaña[3], Marta Mattoso[3]
[3]COPPE/UFRJ
{vviana,kary,marta}@cos.ufrj.br

ABSTRACT
Cloud computing has established itself as a solid computational model that allows for scientists to use a series of distributed virtual resources to execute a wide range of scientific experiments. In several cases, there is a demand for high performance in executing these experiments since many activities are data and computing intensive. Parallelism techniques are a key issue in this experimentation process. There are approaches that provide parallelism capabilities for scientific workflows in clouds. However, most of them rely on the scientist to dimension the virtual cluster to be instantiated. Dimensioning the virtual cluster to execute the workflow in parallel may be a hard task to accomplish, *i.e.* it is hard to define and adapt the optimal number of virtual machines to be used. Most systems follow this manual configuration of the scientist for the whole workflow execution, using adaptive techniques only in the presence of failures. Due to the huge number of options (virtual machine types) to configure a cloud environment, the configuration task commonly becomes impractical to be performed manually, and if it is not adjusted adaptively during the execution, it can impact negatively on workflow performance, or it can produce excessive increase in financial cost. This paper proposes a service called SciDim which is based on the use of a multi-objective cost function allied to genetic algorithms and provenance data to help determining an "ideal" initial configuration for the virtual cluster, under budget and deadline constraints set by the scientist.

Categories and Subject Descriptors
H.4.1: Office Automation - Workflow Management

General Terms
Experimentation, Management, Standardization

Keywords
Cloud Computing, Scientific Workflows, Parallelization

1. INTRODUCTION
Over the last years, for many researchers in the academia, there was a doubt if cloud computing was applicable to the scientific domain [1]. There was also a lot of questioning whether cloud computing was suitable for High Performance Computing (HPC) scientific applications. In early 2010, Juve and Deelman [2] discussed the pros and cons of using cloud computing for scientific workflows. Also in 2010, Oliveira *et al.* [3] surveyed the advantages of cloud computing for e-Science. Later in 2010 [4] we presented SciCumulus, an approach for managing scientific workflow execution on clouds. To improve the use of clouds for scientific workflows, in early 2011 [5], we gave some guidelines on moving scientific experiments to the cloud. Several real bioinformatics workflows [6–9], executed on Amazon EC2, show the potential of cloud computing for scientific workflows, including the ones that need HPC capabilities. In 2012, Malawski *et al.* performed several simulation studies that also evidence the potential usage of clouds [10].

Cloud computing is being adopted by scientists to execute their experiments mainly because it is simpler to allocate virtual resources than acquiring expensive infrastructure elastically. Scientific workflows tend to spend more time being analyzed than executed, generating idle computing time along the experiment life-cycle. This elastic allocation of resources, on demand, provides a new dimension for HPC applications.

Some Scientific Workflow Management Systems (SWfMS) [11,12] bridge the gap between cloud environments and the management of scientific experiments, such as, SciCumulus engine [4,13], Pegasus [14], Swift [15], and specifically for bioinformatics, the combination of Taverna and Galaxy named Tavaxy [16]. In these SWfMS a virtual cluster is created in the cloud to execute the workflow in parallel. Besides managing the parallel execution of the workflow, these SWfMS are responsible for capturing information about the workflow execution, namely provenance [17] and can be used to reproduce and validate a specific experiment.

One fundamental issue considered by these SWfMS that execute workflows in clouds is how to improve performance while not overspending financial resources [10,13]. To solve this issue, existing approaches are primarily focused on optimizing the scheduling plan and task priority, whereas, few approaches evaluate the initial dimensioning the environment as a possible way of optimizing the workflow execution. With the increase in scale of data consumed and produced by scientific workflows, optimizing workflow execution becomes more and more important to improve performance while controlling financial costs during the execution of a scientific workflow.

Native cloud services can scale the amount of virtual machines horizontally and vertically [1]. However, they are general purpose services and do not take into account the specificities of scientific workflows [10]. For example, in the context of scientific workflows we may find several different programs with different behaviors, each with a specific execution time. In addition, in large parameter exploration, this workflow is repeatedly executed and the workflow engine (which is not a general purpose scheduler) has to be aware of the execution pattern of workflow activities, thus considering activity dependencies as well. This way, it is necessary to have a dynamic scheduler for parallel scientific workflows which considers environment dimensioning according to this repeated heterogeneous parameter exploration. Dynamic schedulers for cloud workflows are found in some approaches [10,13,18], however a service for an initial dimensioning of the cloud according to workflow characteristics with budget and deadline constraints set by the scientist is still an open, yet important, problem. Deelman *et al.* [19] discuss this open issue among others in running scientific workflows on clouds.

In this paper, we claim that one way to reduce the total execution time of the workflow while controlling financial costs is to find the initial "best" configuration of the cloud. This initial solution can be further refined by adaptive solutions, but it reduces the unnecessary

use of resources in the cloud right from the start, *i.e.*, to better dimension the virtual cluster to be used in the workflow execution. The main idea is to start using the "optimal" number of virtual machines, or a near optimal value according to the types of virtual machines provided by the cloud service provider. Existing solutions leave the virtual cluster dimensioning task as an "offline" task for scientists. Depending on the cloud provider, there may be several possible combinations to choose and it can be tedious, overpriced and error-prone to be performed manually.

This way, the first step to dimension the virtual cluster is to use a cost function to estimate the behavior of a parallel workflow execution in the cloud. Previously, Viana *et al.* [20] have introduced the modeling of a cost function that takes into account execution time and financial costs to simulate the workflow execution time on a cloud provider. This cost function evolved to the cost function proposed in [13] by Oliveira *et al.* where the authors modeled the cost function associating weights to each criterion. This weighted cost function allows for scientists to fine tune their workflow executions. However, to dimension the virtual cloud cluster it is imperative to use an optimization approach together with a cost function that simulates the workflow execution behavior. In this paper, we have modeled the cost function proposed by Oliveira *et al.* in a fitness function within an optimization technique. We have adopted genetic algorithms [21] as metaheuristics. This optimization is implemented in a component named SciDim (coupled to the SciCumulus workflow engine) for the initial dimensioning of the cloud environment before workflow execution. Using this kind of optimization to dimension the virtual cluster during the workflow execution, as an adaptive execution engine, may not be cost effective, imposing significant overhead to the workflow execution itself. We claim that the cost of this initial optimization, just once, before the workflow execution, can be compensated by leveraging the adaptive execution process and lead to the overall reduction in the workflow execution time.

This paper contributes by introducing a specialized service to help dimensioning the virtual cluster before the workflow execution in the cloud. Although some approaches provide adaptive execution (as SciCumulus engine) that can adjust the amount of resources during execution, the workflow execution performance can be improved if the amount of instantiated virtual machines is already near optimal at the beginning of the execution. This way, in this paper, we present SciDim that aims at dimensioning the virtual cluster before the execution starts. Differently from the current mainstream, it does not explore cost-performance tradeoffs during workflow scheduling. Indeed, the contribution of the paper is optimizing an initial configuration through SciDim, a pre-optimizer component that is based on genetic algorithms and a provenance data gatherer agent. SciDim is evaluated with the SciCumulus adaptive approach, but it can be used before the execution of scientific workflows with other workflow cloud engines, as long as a provenance database can be queried. In fact, SciDim is even more helpful for non-adaptive engines, which have to keep the same initial configuration during the whole workflow execution. Our results show that SciDim obtained a configuration that was near the one obtained by exhaustive search and the time it took to generate the virtual cloud dimensioning, was around 5 min, which is negligible when compared to the gains it can obtain in reducing the total execution time.

This paper is organized into six other sections besides this introduction. Section 2 discusses related work. Section 3 presents SciCumulus, the cloud workflow engine that is coupled to SciDim. Section 4 introduces the workflow formalism necessary for this paper. Section 5 presents SciDim while Section 6 discusses experimental evaluation of SciDim. Finally, Section 7 concludes this paper.

2. RELATED WORK

There are several papers that provide multi-objective scheduling in clusters and grids as discussed in [10]. However, in the cloud computing scenario there are a few papers, such as Malawski *et al.* [10], Janssens *et al.* [22] and Tian [23], that provide runtime dimensioning of cloud environments for several different purposes. In this way, we provide a brief description of these papers that are related in some way to the initial dimensioning that we are addressing in this paper.

Malawski *et al.* [10] propose a series of adaptive scheduling algorithms for cloud-based scientific workflows. The main idea is to scale the amount of virtual machines during workflow execution to comply with scientists' constraints. Performance results presented by Malawski *et al.* [10] are based on simulations of several workflow executions. We believe that the performance of these adaptive virtual machine configurations can be improved if the virtual cluster dimensioning is optimized before the workflow execution.

Janssens *et al.* [22] propose a component to implement short-term call load predictions and combine this with history-based predictions to anticipate future call load changes and use these predictions to dimension Session Initiation Protocol (SIP) services for communications. Although Janssens *et al.* do not dimension a virtual cluster they give important driving guidelines about dimensioning cloud services for business applications. However, they do not take into account several criteria as financial cost or reliability when dimensioning SIP services.

Tian [23] proposes a technique called adaptive dimensioning for cloud data centers that is based on collecting statistics from past accesses (part of what we consider as provenance data in scientific workflows) so that the right amount of computing resources can be allocated for variable workloads to meet quality of service requirements. This dimensioning is adjusted "on the fly" while new statistics are collected. Although the work of Tian is a step forward, it cannot be used in the scientific domain since it does not consider the data dependency between programs as the case of scientific workflows.

Finally, although these approaches represent a step forward, they leave room for the optimization of the initial dimensioning of the environment, *i.e.*, brings opportunities for adjusting the amount virtual machines before the execution of scientific workflow, according to the constraints defined by the scientists. This may speedup the dynamic convergence of the run-time approaches.

3. SCICUMULUS WORKFLOW ENGINE

This section presents an overview of the SciCumulus cloud workflow engine and its provenance schema. SciCumulus is the chosen workflow engine to be associated with SciDim for the experiments presented in this paper. SciCumulus is an important component of this paper since it provides an adaptive scheduler for parallel scientific workflows to be combined with SciDim. It is important to highlight that other cloud workflow execution engines, besides SciCumulus, can also benefit from this initial cloud dimensioning, particularly non-adaptive approaches.

3.1 Architectural Overview

SciCumulus is an engine that manages the parallel execution of scientific workflow activities in a cloud, such as Amazon EC2 [24] or GoGrid [25]. SciCumulus has four tiers: Client tier, Distribution tier, Execution tier and Data tier. A high-level conceptual architecture is summarized in Figure 1. SciCumulus is itself

distributed and its tiers are part of the workflow engine. To find more information about SciCumulus components, please refer to Oliveira *et al.* [4].

SciCumulus client tier is responsible for initiating the execution of workflow activities in the cloud. The components of this tier are deployed in a SWfMS, such as VisTrails [26] or Kepler [27]. Components installed in the SWfMS upload and download data and start the parallel execution by invoking the distribution tier. The distribution tier manages the execution of parallel activities in clouds by creating cloud activities (*i.e.* the parallel execution of a workflow activity in the cloud – more information can be obtained in [28]) that contain the program to be executed and parameters values. This tier is responsible for creating the scheduling plan for the cloud activities to be executed in the different virtual machines that are part of the execution tier. The execution tier is responsible for invoking executable codes (*i.e.* programs that are part of the workflow) in many virtual machines available for use. Besides executing the workflow programs, the execution tier collects and stores provenance data in the repositories that are part of the data tier. Finally, the data tier has the provenance repository and the file system that stores all files produced and consumed during workflow execution. With these four tiers, SciCumulus provides a computational infrastructure to support workflow parallelism with provenance gathering.

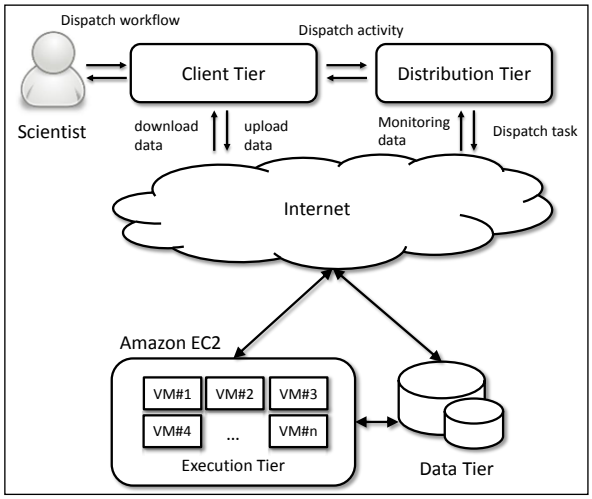

Figure 1 SciCumulus as proposed by Oliveira *et al.* [13]

Differently from other SWfMS and parallel implementations such as Hadoop based [29], SciCumulus is able to execute either in static or adaptive [13] execution modes. In the static mode, it generates a scheduling plan based on a static scenario of the environment at the beginning of the execution, *i.e.* it only considers the amount of virtual machines and the state of all virtual machines in the beginning of a workflow execution.

However, there may be several performance fluctuations in the cloud during the course of the experiment, such as virtual machine failures or system updates. These variations may generate severe performance degradation on the execution. Thus, in the adaptive mode, SciCumulus analyzes the real processing power of each virtual machine by querying provenance data, and the performance restrictions (*i.e.* maximum execution time and maximum financial cost, for example). Then it dynamically scales the number of virtual machines used in the execution up and down according to the changes and availability of resources of the cloud environment, aiming at maintaining the overall performance of the workflow. This scaling is horizontal where SciCumulus scales by adding more machines into the existing pull of resources. Although the adaptive mode is a step forward it demands a certain quantum of time to set the ideal (or near the ideal) number of virtual machines during execution.

If the number of virtual machine is the ideal or near the ideal (the amount of virtual machines that provides the best performance or the minimum financial cost following scientists' criteria) in the beginning of the execution, it could spare significant amount of time. This way, SciDim empowers SciCumulus adaptive approach. Since the main focus of this paper is SciDim, we only present the adaptive approach in high level of abstraction.

3.2 Provenance Model

SciCumulus uses PROV-Wf model [30] to represent information about the workflow being executed and about the cloud environment. PROV-Wf is an specialization of PROV (a W3C recommendation) [31]. It is designed to represent causal relations between activities, agents, entities and roles involved in a parallel workflow execution. The provenance model is composed of three main parts: the structure of the workflow, execution of the workflow and the environment configuration. The components of the provenance model are identified and stereotyped with the types of the PROV ontology [31]. A scientific workflow (*i.e.* entity "Workflow") is composed of a set of activities (*i.e.* entity "Activity"). Each activity is responsible for executing a specific program in a machine with a specific configuration. One invocation of a program (*i.e.* cloud activity) within a workflow consumes a set of parameters values. To express all data that is consumed and produced by execution instances, the entity "RelationSchema" is associated with a schema and can be defined with multiple fields. Each field (*i.e.* entity "Field") describes the meaning of each parameter associated to a program that is associated to an activity. The entity "Value" expresses the set of a specific field values, each set associated to a specific execution instance. Furthermore, the entity "File" represents all files consumed and produced by a workflow execution. "Machine" and "Program" are considered agents because they act on the workflow execution on behalf of the scientist. Furthermore, the activities in PROV data model are mapped to describe an action that happens in a period of time (during workflow execution). "Execute Activity" is responsible to represent the execution of some activity instance, while "Execute Workflow" defines the execution properties of one SWfMS. Also, some activities are defined to represent the consumption and production of data (*e.g.* the activity Use field). The PROV model can be helpful for dimensioning the virtual cluster, but PROV-Wf adds to PROV details on execution information of the workflow, such as start time and end time of each activity instance, properties of one SWfMS and machine configurations of the execution environment.

4. WORKFLOW FORMALISM

The formalism presented for representing workflows and cloud is the same proposed by Oliveira *et al.* [13] and Ogasawara *et al.* [32]. This section summarizes the workflow formalism used in this paper. We consider a scientific workflow as a Directed Acyclic Graph (DAG) named *W(A,Dep)*. The set $A = \{a_1, a_2, \ldots, a_n\}$ represents all activities that compose *W* while the set *Dep* represent the edges associated to data dependency between activities in *A*. Given $a_i \mid (1 \leq i \leq n)$, *let* $I = \{i_1, i_2, .., i_m\}$ be the input dataset for activity a_i, then $Input(a_i) \subset I$. Also, let *O* be the output data produced by a_i, then $Output(a_i) \subset O$. A specific activity in *A* is modeled as *a(time)* where *time* is the total execution time of *a*.

The dependency between two activities is modeled as $dep(a_i, a_j, ds) \leftrightarrow \exists\, O_k \in Input(a_j) \mid O_k \in Output(a_i)$ and *ds* is the size of data transferred between these two activities. Although the data size *ds* does not vary for the different workflow executions we have to

explicitly represent it in order to use in the cost function. Given a workflow W, a set $Ca = \{ca_1, ca_2, ..., ca_k\}$ of cloud activities [7] (a synonym of parallel task in the context of this paper) is created for its execution. Each cloud activity ca_i is associated to a specific a_i which is represented as $Act(ca_i) = a_i$. A set of cloud activities of a workflow activity a_i is denoted as $Ca(a_i)$. Each ca_i consumes its own input data $InputCa(ca_i)$ and produces output data $OutputCa(ca_i)$. We establish the dependency between two cloud activities as $DepCa(ca_i, ca_j, ds) \leftrightarrow \exists\ r \in Input(ca_j) \mid r \in Output(ca_i) \wedge dep(Act(ca_i), Act(ca_j), ds)$. A specific cloud activity ca_i is modeled as $ca(time)$ where $time$ is the total execution time of ca. Also consider $VM = \{VM_1, VM_2,.., VM_m\}$ as the set of m virtual machines that composes a virtual cluster and that is used for executing the scientific workflow W in parallel. Each VM_j is associated to a specific computational slowdown index (csi) as proposed by Boeres *et al.* [33]. The $csi(VM)$ function is used since each virtual machine of the virtual cluster is classified according to its processing power. This metric is designed to be inversely proportional to the computational power of VM_j.

5. CLOUD INITIAL DIMENSIONING

Several computing resources are available for use in different cloud environments. For example, Amazon EC2 alone provides more than 10 different types of virtual machines to be chosen. This way, selecting which type and the amount of virtual machines to instantiate to run a specific workflow is a complex decision. If the amount of virtual machines is under or over dimensioned it surely produces negative impacts on the performance of the workflow or the financial costs of the execution. To avoid under and over dimensioning of the cloud environment, we propose the use of a specific service named SciDim that dimensions the cloud environment (*i.e.* the virtual cluster) according to scientists' constraints, such as maximum execution time and maximum financial cost. SciDim is especially helpful for cloud execution engines.

SWfMS works with static and adaptive scheduling modes in clouds. In both scheduling modes, it is important that the initial set of instantiated virtual machines is ideal or close to the ideal. In the case of static scheduling, this problem becomes more evident as the machines available at the start of execution are the same until the end of execution. In the case of adaptive scheduling, although this scheduling mode is able to horizontally scale up and down the amount of virtual machines involved in the execution, this adaptability may introduce a significant processing overhead. SciDim is composed of an autonomous provenance gatherer agent, a workflow execution cost function and an optimizer component and it is invoked before the execution of the scientific workflow. The agent captures information from the cloud environment independent of the workflow executions and stores it in a provenance repository. The cost function is used to evaluate the cloud dimensioning and optimizer searches the solution space subjected to the constraints defined by the scientist.

5.1 Provenance Gatherer Agent

In clusters and computational grids, there is an initial financial investment (sometimes large) and an operating cost paid over time (cost of energy consumption, for example) and these factors do not impact the scheduling of tasks (*i.e.* the scheduler does not take into account these characteristics when generating the scheduling plan for a scientific workflow since this cost is already amortized). In clouds there is no initial investment since resources are acquired on demand, based on a pay *per* use model. Thus, if the dimensioning of the virtual cluster is poorly performed, it can generate additional financial costs to scientists.

To predict the execution cost for an activity, the agent needs information about the various cloud providers and services available in the cloud. In SciDim, this information is collected via an autonomous agent, called Provenance Gatherer, capable of capturing data about hardware, software, and providers themselves. Some of these data is related to the types of virtual machines available, their price, type of storage, operating systems, and transfer fees, among other informations.

The agent starts working from the moment the scientist selects one or more cloud service providers, as eligible, to provide the virtual infrastructure to execute a scientific workflow. Thereafter, the agent (using APIs provided by cloud providers) captures the required data. When the agent starts working, it creates new threads to process the request for provenance information that was delegated to it. First of all, each thread extracts specific information about the cloud environment and the cloud provider (pricing virtual machine types, etc.). After that, if this provenance extraction succeeds, the agent checks if this information was previously captured by querying the workflow provenance database. If one step of this provenance extraction steps fails, the agent generates a log entry and then closes the communication to allow all threads to terminate. In its current version, the agent contemplates only Amazon EC2 provider, which was chosen due to its stability and reliability. All collected data are stored in a relational database provenance instantiated in PostgreSQL, which is also an instance in the cloud. The database used by the agent follows the PROV-Wf model previously presented.

5.2 3-Objective Cost Function

The cost function used in SciDim is the 3-Objective weighted cost function proposed by Oliveira *et al.* [13]. This cost function is implemented in SciCumulus to estimate costs involved on executing a cloud activity in a particular cloud environment. In this paper, the cost function is used to simulate the entire execution of the workflow to dimension the cloud environment.

Here follows a summary of the cost function used in SciDim, more details can be found in [13]. To model the total execution time, let us consider $P(ca_i, VM_j)$ as the expected execution time of a cloud activity ca_i in a specific VM_j, this way $P(ca_i, VM_j)$ can be represented as $ca_i.time \times csi(VM_j)$. The value of $P(ca_i, VM_j)$ has to be estimated to use this cost function. This estimation can be performed using information of previous executions of workflows by querying the PROV-Wf schema on the provenance repository of SciCumulus. When there is available data of previous executions the average execution time is calculated and this information is used as input to the cost function. Since SciDim gets workflow execution information from SciCumulus PROV-Wf, it can be adapted to get provenance data from other SWfMS that are also PROV compliant.

Besides the execution time of a cloud activity, it considers the reliability of the execution. Virtual machines commonly present performance fluctuations and failures during workflow execution. It considers that virtual machine failures follow a Poisson distribution $F(VM_j)$, $\forall\ VM_j \in VM$ with constant value. Consequently $F(VM_j)$ expresses the probability of a given number of failures in VM_j happen during workflow execution independently of the last virtual machine failure. Reliability criterion called $R(ca_i,VM_j)$ can defined as: $F(VM_j) \times P(ca_i,VM_j)$.

The third criterion is financial cost. Financial cost in clouds depends on the execution time of time those virtual machines. In this paper, the total execution time is defined as a sum of several quanta of time Qt which generated the set $\delta = \{\delta_1, \delta_2, ..., \delta_x\}$ of quanta. For each VM_j involved in the execution we have to pay a value per quantum (hour, minute) that we named Vq. Let us

consider $\omega(\delta_i, VM_j)$ as a function to return if the VM_j is executing a cloud activity in the quantum δ_i. Thus, the total execution financial cost for a schedule φ is defined as following:

$$M(\varphi) = Vq \times \sum_{j=1}^{|VM|} \sum_{j=1}^{MS/Qt} \omega(\delta_i, VM_j)$$

Note that in the original cost function, the financial cost is not a single entity as we present here. It considers data transfer costs, storage costs and others. However, in this paper we simplified the financial cost estimation since transfer and storage costs can be considered negligible when compared to the execution time costs of computing intensive workflows.

To define a schedule based on the three objectives: total execution time, financial cost and reliability, we use a weighted cost function to set the weight for each criterion. For each VM_j in the set VM that is idle and requests for a cloud activity, it is then performed a search for the best ca_i in the list of available cloud activities (named Ca') to execute in VM_j following the 3-Objective cost function. Given $VM_j \in$ VM as the next virtual machine to execute a cloud activity, the optimizer has to find $ca_i \in Ca'$ which minimizes the following cost function:

$$f(ca_i, VM_j) = \alpha_1 \times P(ca_i, VM_j) + \alpha_2 \times R(ca_i, VM_j) + \alpha_2 \times M(\varphi)$$

By allowing scientists to inform the three alpha values, it is possible to fine tune priorities in the workflow execution parameters. The cost function seen in this section is used by the optimizer of SciDim to define a set of virtual machines near to ideal for the execution of a scientific workflow in a cloud.

5.3 Optimizer Component

SciDim has an optimizer component that is responsible for evaluating several configurations of the environment to choose the most suitable for a specific scientific workflow and its associated execution criteria (execution deadline and financial cost limits). This way, the optimizer uses the cost function to simulate the execution and the scheduling of the workflow under different configurations. As mentioned earlier, the main objective is to inform SciCumulus what types of virtual machines and how many virtual machines of each type must be instantiated in the environment, as near as possible to the ideal amount of virtual machines. In other words, this service is responsible for optimizing the selection and dynamic instantiation of virtual machines before scheduling workflow activities. The choice of virtual machines types to instantiate resembles the traveling salesman problem [34], where there are many paths that must be traversed with the lowest total cost, as we have several possible configurations and one must be chosen that yields the lowest cost.

Thus, since it is difficult (if not impractical) to determine the optimal solution of this problem in an acceptable time, we chose to use a method based on heuristics [35]. We use Genetic Algorithms [36] for choosing the set of virtual machines to be instantiated. The selection by genetic algorithms is due to the fact that they are relatively simple and fast algorithms, besides being indicated for cases where the search space is large and the problem does not require the optimal solution.

To implement the optimizer we used JGAP framework [37], which provides basic mechanisms that can be easily used to apply evolutionary principles to the problem to be solved. Initially, we need to set a configuration object describing how we want our chromosomes to be configured. The chromosome is a potential solution and is divided into multiple genes. The first step is to decide how many genes are necessary and what each gene represents. In SciDim, the chromosomes are composed by four genes, each one associated to a specific type of virtual machine (since we follow Amazon EC2 we only consider micro, large, extra-large and extra-large high capacity virtual machine types). Gene values (*i.e.* alleles) are integer values, since they represent the amount of virtual machines of this type to be instantiated. Scientists have to specify a lower and upper limit, which define the values for each type of virtual machine (if we consider Amazon EC2 the minimum value is 0 and the maximum is 20).

The second step is to develop a fitness function. The fitness function is a unique method that is modeled (according to the problem being solved), which returns an integer value that indicates how good is the solution found when compared to other generated solutions. In the case of this paper, the fitness function evaluates if a specific configuration led to a cloud activity scheduling that complies with scientists' constraints. This way, we used a single function based on the presented cost function where we considered all costs involved in all schedules generated:

$$g(x) = 1 - \left\lceil \frac{(\sum_{i=1}^{n} f(cai, vmj))}{|CA|} \right\rceil$$

The value of $g(x)$ has to be compared with the constraints set by the scientist for deadline and budget. If the generated value complies with these constraints it can be considered for evaluation. Note that this value is always smaller than 1 and the algorithm chooses the configuration that provides the higher value and complies with scientists constraints. During the process of evolution, chromosomes are exposed to various operators such as mutation and then are selected for the next generation based on the fitness value. The fitness function was developed considering that all criteria of the cost function are taken into account. In each generation, we performed a simulation of the entire workflow execution based on a simulator [20] that implements the cost function.

The third step is to indicate how many chromosomes exist in population during the execution of the genetic algorithm. We used standard values widely used in similar problems in the literature. The initial population of chromosomes is generated randomly by a JGAP method: 100 individuals. JGAP was configured to perform 200 generations since we cannot consume a long time in the environment configuration instead of using this time for workflow execution. Besides we set to have crossing rate of 35% of the population and an 8% rate of mutation of individuals, which are default values for JGAP. After these adjustments, the algorithm is ready to begin the evolution of population and return the best configuration found after simulation. Although we achieved satisfactory solutions using this configuration, new studies have to be performed to analyze parameter's influence on the quality of the solution and the time needed to find the best solution.

6. EXPERIMENTS AND EVALUATION

In this section, we present the evaluation of SciDim. To analyze the feasibility of the proposed approach, we use the bioinformatics workflow SciPhy for phylogenetic analysis [6]. This workflow is detailed throughout this section. The main objective of this section is to analyze the performance of SciCumulus with and without using SciDim to dimension the virtual cluster. To better evaluate, we used SciPhy scientific workflow, a real workflow in the bioinformatics domain. Several tests were performed to evaluate SciDim. These tests were based on previous parallel executions of workflows where the best environment configuration was given.

6.1 Environment Setup

For the experiments performed in this paper, we executed SciCumulus coupled to SciDim in Amazon EC2 cloud. Amazon EC2 offers several different types of virtual machines for general

use. In the experiments presented in this paper we have just considered Amazon's Micro type (EC2 ID: t1.micro - 613 MB RAM, 1 core, EBS storage volume of 30 GB); the Large type (EC2 ID: m1.large - 7.5 GB RAM, 2 cores, 850 GB storage); Extra Large type (EC2 ID: m1.xlarge - 15 GB RAM, 4 cores, 1690 GB of storage); and High-CPU Extra Large Instance type (EC2 ID: c1 . xlarge - 7.5 GB RAM, 8 cores, 850 GB storage) .

6.2 Workflow SciPhy

To perform the experiments to evaluate feasibility of SciDim, let us consider the following real scientific workflow from the bioinformatics domain named SciPhy. Phylogenetic workflows aim at producing phylogenetic trees to support the existing evolutionary relationships that are used by specialists for inferring a phylogeny. SciPhy [6] is one example of phylogenetic analysis workflow. SciPhy aims at processing a large collection of multi-fasta files to obtain phylogenetic trees, assisting biologists to explore phylogeny and to determine the evolutionary life of genes or genomes. Typically, one execution of SciPhy in parallel on clouds consuming 50 multi-fasta files generates 1,250 parallel activities demanding 4.19 days using 16 virtual cores. SciPhy consists of seven main activities: (i) construction of the multiple sequence alignment (MSA); (ii) conversion of the MSA format; (iii) search for the best evolutionary model; (iv) construction of the phylogenetic tree; (v) concatenation of the alignments produced to generate a superalignment; (vi) election of the best evolutionary model for the superalignment; and (vii) construction of the phylogenomic tree. This phylogenomic tree helps to infer the evolutionary and phylogenetic relationships of organisms included in the study.

6.3 Calibrating the Optimizer

Before evaluating SciDim while using SciPhy we have performed a calibration of the cost function and the optimizer with a different workflow so that results obtained in this paper would not be biased to SciPhy. As a real case we considered 524 previous executions of X-ray crystallography workflow [7], which was also executed in parallel on the Amazon EC2 cloud. The workflow executions used to calibrate the model refer to the parallel processing of 2,000 input images separated into groups of 3 images, generating a set of 667 activities to be processed in parallel. This workflow lasts for approximately 12.8 hours when executed in 2 cores. Starting from an initial scenario (no virtual machine created), the optimizer executes the genetic algorithm to determine the number of virtual machines to instantiate. For each chromosome it checks if the value of the fitness function complies with scientist's constraints. The evolutions continue until reaching the final solution according to the constraints imposed by the scientist. In this calibration, the simulations are always performed using only one virtual machine type. Figure 2 shows the results of the execution simulation time for the settings chosen by the optimizer, and it also presents the real execution time of SciCumulus. We can state that the simulated results presented a similar behavior to the real results which indicate that the cost function and the optimizer are not biased. Figure 3 presents the comparison between real financial costs and simulated financial costs. As we can see, there is a small time difference between the simulation in SciDim and real in SciCumulus. This shows that the cost function provides over estimated prediction. This difference is adequate since scientists can set an upper bound for financial costs.

In the experiments, the constraints were US$ 10.00 of financial cost and 1 hour maximum execution time. In this simulation, the time limit imposed was achieved using between 16 and 256 virtual cores, as shown in Figure 2. Although the performance was equivalent using different types of virtual machines, the financial cost was significantly different. For example, the value paid in the simulations varied from US$ 2.44 (16 cores) to US$ 39.16 (256 cores). Thus, the optimizer determined the optimum number of virtual machines as 16 of large type which was the same configuration achieved using a brute-force search. The alpha values in the cost function were $\alpha_1 = 0.10$, $\alpha_2 = 0.10$ and $\alpha_3 = 0.90$ to prioritize the financial cost.

6.4 Genetic Algorithm × Brute-force Search

To measure the distance of the execution time between genetic algorithms and brute force search in the SciPhy workflow scenario, we compared the two for combinations of virtual machines until 64 virtual cores. We considered now the SciPhy workflow processing 400 multi-fasta files and limiting the financial budget at US$ 40.00. We analyzed several configuration possibilities of the environment, *i.e.*, 2, 4, 8, 16, 32, and 64 virtual machines. Although Amazon EC2 only allows for the instantiation of 20 virtual machines, these tests were performed without taking into account this limitation (using more than one user account), since this limitation is not found among all providers of this type of service. Genetic algorithms were executed with 200 evolutions, with an initial population of 100 chromosomes.

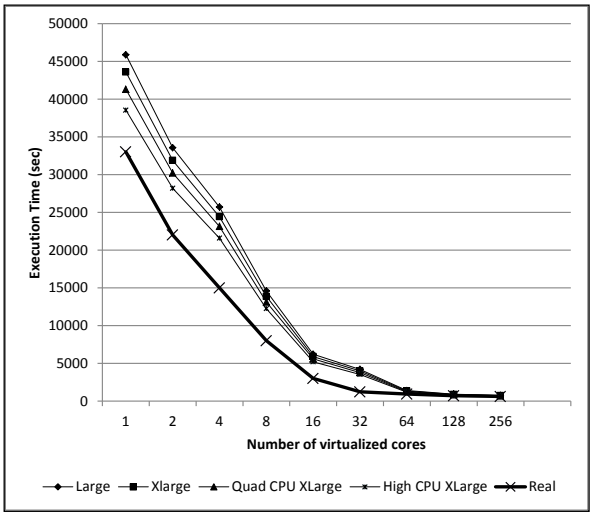

Figure 2 Simulated *versus* real results for X-ray workflow

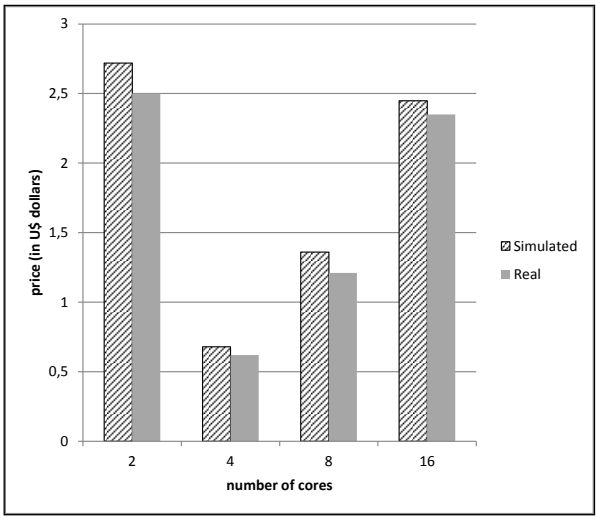

Figure 3 Simulated versus real financial cost

As expected, the necessary amount of time to find the optimal configuration becomes prohibitive as the number of virtual machines involved in the execution increases. As presented in

Figure 4, the genetic algorithm maintains almost the same time to optimize the configuration of the environment, regardless of the number of virtual machines that must be evaluated. This happens because the time to execute the genetic algorithm is based only on chromosome population size and number of evolutions it is programmed to perform. It is noteworthy that the greater the number of evolutions, the higher the quality of the optimization. As expected, the optimization time of the algorithm that evaluates all possible cases (brute force search) increases exponentially, as the number of virtual machines increases.

6.5 SciDim Performance Analysis

To assess whether SciDim improved the performance of SciPhy executing the adaptive method of SciCumulus, we first show the behavior of SciCumulus without SciDim. When executing adaptive scheduling, SciCumulus distributes all activities according to existing resources and it is able to scale the amount of resources involved in the execution up and down (during workflow execution). Although the adaptive approach adjusts the amount of resources during execution, these adjustments impose some overhead that can impact on the performance of the execution. An example of the use of SciDim with the SciCumulus is presented in Figure 5 where we show three curves: using SciDim, using SciCumulus adaptive approach without SciDim and using the ideal configuration obtained by brute force search.

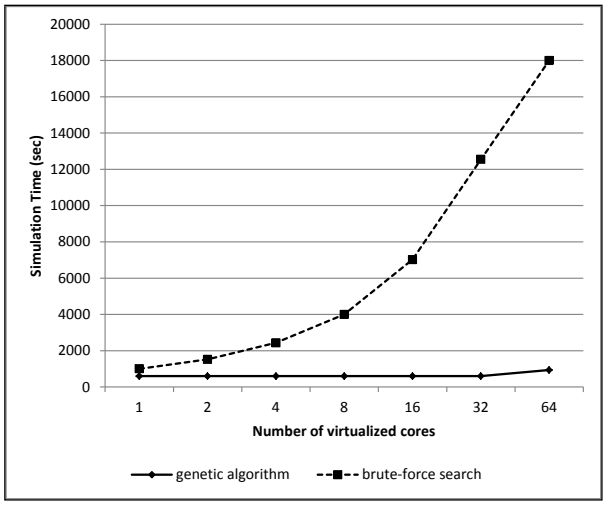

Figure 4 Genetic Algorithms *versus* brute-force search

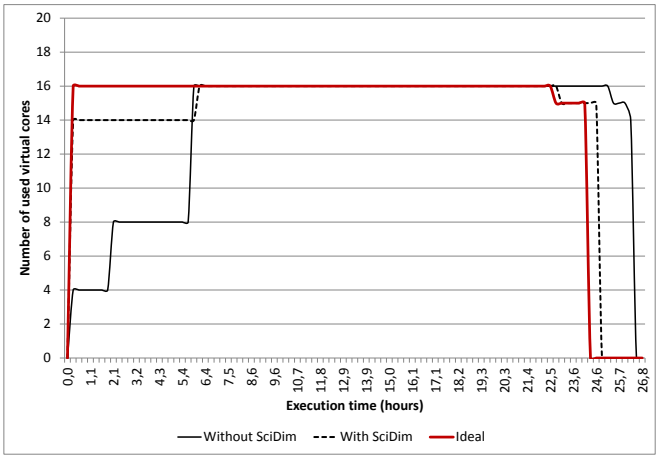

Figure 5 SciCumulus executing with and without SciDim

In Figure 5, we executed the SciPhy workflow processing 400 multi-fasta files and limiting the financial budget at US$ 75.00. Without SciDim, SciCumulus started execution with no virtual machines instantiated and it scales the number of machines as it analyzes data provenance that are generated at runtime. In this experiment, the ideal number of virtual machines is 16 virtual machines (we know this by executing a brute force search). Using SciDim we already started executing the workflow using 14 virtual machines (the number informed by the genetic algorithm execution) and this way only 2 new virtual machines were instantiated during workflow execution. When executing SciCumulus without SciDim we have to instantiate 16 new virtual machines during execution. Due to this difference, there is a 7% total execution time difference between running SciPhy using SciCumulus coupled to SciDim and without using SciDim, meaning approximately 2 more hours to finish without SciDim. This difference tends to increase when we execute more sophisticated workflows, *i.e.*, the longer the execution time of the workflow, the greater will be the absolute time difference. When comparing the results with SciDim with the ideal result we can state a difference of 3% in the total execution time. This difference is acceptable since the cost of optimizing the virtual cluster size before workflow execution was negligible when compared to the workflow execution. In all experiments, we performed 200 evolutions in the genetic algorithm, with an initial population of 100 chromosomes; the SciDim needed 5.5 minutes in average to estimate the amount of virtual machines to instantiate. This time can be considered negligible compared to the total execution time of the workflow and the performance improvements SciDim obtained.

7. CONCLUSIONS AND FINAL REMARKS

This paper presents SciDim, a service used for dimensioning the amount of resources to allocate for a parallel workflow execution. This service is composed by a cost function and an optimizer component that estimates costs based on three criteria: total execution time, reliability and financial cost of the workflow execution. Allied with the cost function, the optimizer component presented in this paper was responsible for determining the best configuration for the environment. The result is the amount of virtual machines for each type, available in the provider, which should be instantiated for the workflow execution subject to constraints set by the scientists. The benefits from investing on an initial solution were evaluated in a series of experiments. SciDim was able to reduce the total execution time of a real workflow in SciCumulus in 2 hours, by taking 5 minutes to generate the initial dimensioning of the virtual cluster before the workflow execution. As future work, we plan to improve on SciDim by analyzing the optimality of results from the genetic algorithms as well as other metaheuristics.

8. ACKNOWLEDGMENTS

The authors would like to thank CNPq, CAPES and FAPERJ.

9. REFERENCES

[1] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, 2009, A break in the clouds: towards a cloud definition, *SIGCOMM Comput. Commun. Rev.*, v. 39, n. 1, p. 50–55.

[2] G. Juve and E. Deelman, 2010, Scientific workflows and clouds, *Crossroads*, v. 16 (mar.), p. 14–18.

[3] D. Oliveira, F. Baião, and M. Mattoso, 2010, "Towards a Taxonomy for Cloud Computing from an e-Science Perspective", *Cloud Computing: Principles, Systems and Applications*, Nick Antonopoulos and Lee Gillam edHeidelberg: Springer-Verlag

[4] D. Oliveira, E. Ogasawara, F. Baião, and M. Mattoso, 2010, SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific

Workflows, In: *3rd International Conference on Cloud Computing*, p. 378–385, Washington, DC, USA.

[5] D. Oliveira, F. Baiao, and M. Mattoso, 2011, Migrating Scientific Experiments to the Cloud, *HPC in the Cloud* (mar.)

[6] K.A.C.S. Ocaña, D. Oliveira, E. Ogasawara, A.M.R. Dávila, A.A.B. Lima, and M. Mattoso, 2011, "SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes", In: *Advances in Bioinformatics and Computational Biology*, , chapter 6832, Springer, p. 66–70.

[7] D. Oliveira, K. Ocana, E. Ogasawara, J. Dias, F. Baiao, and M. Mattoso, 2011, A Performance Evaluation of X-Ray Crystallography Scientific Workflow Using SciCumulus, In: *IEEE International Conference on Cloud Computing (CLOUD)IEEE Cloud*, p. 708–715, Washington, D.C., USA.

[8] A. Matsunaga, M. Tsugawa, and J. Fortes, 2008, CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications, *IEEE eScience 2008*, p. 222–229.

[9] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, 2008, On the use of cloud computing for scientific workflows, In: *IEEE Fourth International Conference on eScience (eScience 2008)*, p. 7–12, Indianapolis, USA.

[10] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, 2012, Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 22:1–22:11, Los Alamitos, CA, USA.

[11] E. Deelman, D. Gannon, M. Shields, and I. Taylor, 2009, Workflows and e-Science: An overview of workflow system features and capabilities, *Future Generation Computer Systems*, v. 25, n. 5, p. 528–540.

[12] I.J. Taylor, E. Deelman, D.B. Gannon, and M. Shields, 2007, *Workflows for e-Science: Scientific Workflows for Grids*. 1 ed. Springer.

[13] D. Oliveira, K. Ocaña, F. Baião, and M. Mattoso, 2012, A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds, *Journal of Grid Computing*, v. 10, n. 3, p. 521–552.

[14] G. Juve, E. Deelman, G.B. Berriman, B.P. Berman, and P. Maechling, 2012, An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2, *Journal of Grid Computing*, v. 10, n. 1 (mar.), p. 5–21.

[15] M. Wilde, M. Hategan, J.M. Wozniak, B. Clifford, D.S. Katz, and I. Foster, 2011, Swift: A language for distributed parallel scripting, *Parallel Computing*, n. 37(9), p. 633–652.

[16] M. Abouelhoda, S. Issa, and M. Ghanem, 2012, Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support, *BMC Bioinformatics*, v. 13, p. 77.

[17] J. Freire, D. Koop, E. Santos, and C.T. Silva, 2008, Provenance for Computational Tasks: A Survey, *Computing in Science and Engineering, v.10*, n. 3, p. 11–21.

[18] A. Jeongseob, K. Changdae, and J. Han, 2012, Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources, In: *Proceedings of the HotCloud '12*, Boston, MA.

[19] E. Deelman, G. Juve, and B. Berriman, 2012, Using clous for science, is it just kicking the can down the road?, In: *CLOSER 2012*, Portugal.

[20] V. Viana, D. de Oliveira, and M. Mattoso, 2011, Towards a Cost Model for Scheduling Scientific Workflows Activities in Cloud Environments, In: *2011 IEEE World Congress on Services (SERVICES)*, p. 216–219

[21] L.D. Davis, 1991, *Handbook Of Genetic Algorithms*. 1st ed. Van Nostrand Reinhold.

[22] N. Janssens, X. An, K. Daenen, and C. Forlivesi, 2012, Dynamic scaling of call-stateful SIP services in the cloud, In: *Proceedings of the 11th international IFIP TC 6 conference on Networking - Volume Part I*, p. 175–189, Berlin, Heidelberg.

[23] W. Tian, 2009, Adaptive Dimensioning of Cloud Data Centers, In: *Proceedings of the 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, p. 5–10, Washington, DC, USA.

[24] Amazon EC2, 2010, *Amazon Elastic Compute Cloud (Amazon EC2)*, http://aws.amazon.com/ec2/.

[25] GoGrid, 2012, *Cloud Hosting, Cloud Servers, Hybrid Hosting, Cloud Infrastructure from GoGrid* URL: http://www.gogrid.com/.

[26] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, and H.T. Vo, 2006, VisTrails: visualization meets data management, In: *SIGMOD*, p. 745–747, Chicago, Illinois, USA.

[27] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, 2004, Kepler: an extensible system for design and execution of scientific workflows, In: *Scientific and Statistical Database Management*, p. 423–424, Greece.

[28] D. Oliveira, E. Ogasawara, K. Ocana, F. Baiao, and M. Mattoso, 2011, An Adaptive Parallel Execution Strategy for Cloud-based Scientific Workflows, *Concurrency and Computation: Practice and Experience*, v. 24, p. 1531-1550

[29] Apache Software Foundation, 2009, Hadoop, *Apache Hadoop Website*

[30] F. Costa, V. Silva, D. Oliveira, K. Ocana, J. Dias, E. Ogasawara, and M. Mattoso, 2013, Capturing and Querying Workflow Runtime Provenance with PROV: a Practical Approach, In: *BigProv'13*, EDBT/ICDT Workshops 2013, Genova, Italy.p.282-289

[31] L. Moreau and P. Missier, 2011. The PROV Data Model and Abstract Syntax Notation. *W3C Working Draft. (Work in progress.)*. URL: http://www.w3.org/TR/prov-dm/.

[32] E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, and M. Mattoso, 2011, An Algebraic Approach for Data-Centric Scientific Workflows, *Proc. of VLDB Endowment*, v. 4, n. 12, p. 1328–1339.

[33] C. Boeres, I. Sardiña, and L. Drummond, 2011, An efficient weighted bi-objective scheduling algorithm for heterogeneous systems, *Parallel Computing*, v. 37, n. 8 (ago.), p. 349–364.

[34] D.L. Applegate, R.E. Bixby, V. Chvatal, and W.J. Cook, 2007, *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

[35] T. Bäck, 1996, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford, UK, Oxford University Press.

[36] D.E. Goldberg, 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*. 1 ed. Addison-Wesley Professional.

[37] K. et al. Meffert, 2012. JGAP - Java Genetic Algorithms and Genetic Programming Package. URL: http://jgap.sf.net..