J Grid Computing (2012) 10:521–552
DOI 10.1007/s10723-012-9227-2

A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds

Daniel de Oliveira · Kary A. C. S. Ocaña · Fernanda Baião · Marta Mattoso

Received: 30 September 2011 / Accepted: 9 August 2012 / Published online: 25 August 2012

Abstract In the last years, scientific workflows have emerged as a fundamental abstraction for structuring and executing scientific experiments in computational environments. Scientific workflows are becoming increasingly complex and more demanding in terms of computational resources, thus requiring the usage of parallel techniques and high performance computing (HPC) environments. Meanwhile, clouds have emerged as a new paradigm where resources are virtualized and provided on demand. By using clouds, scientists have expanded beyond single parallel computers to hundreds or even thousands of virtual machines. Although the initial focus of clouds was to provide high throughput computing, clouds are already being used to provide an HPC environment where elastic resources can be instantiated on demand during the course of a scientific workflow. However, this model also raises many open, yet important, challenges such as scheduling workflow activities. Scheduling parallel scientific workflows in the cloud is a very complex task since we have to take into account many different criteria and to explore the elasticity characteristic for optimizing workflow execution. In this paper, we introduce an adaptive scheduling heuristic for parallel execution of scientific workflows in the cloud that is based on three criteria: total execution time (makespan), reliability and financial cost. Besides scheduling workflow activities based on a 3-objective cost model, this approach also scales resources up and down according to the restrictions imposed by scientists before workflow execution. This tuning is based on provenance data captured and queried at runtime. We conducted a thorough validation of our approach using a real bioinformatics workflow. The experiments were performed in SciCumulus, a cloud workflow engine for managing scientific workflow execution.

Keywords Cloud computing · Scientific workflow · Scientific experiment · Provenance

D. de Oliveira (✉) · K. A. C. S. Ocaña · M. Mattoso
Federal University of Rio de Janeiro - COPPE/UFRJ,
P.O. Box 68511, 21941-972 Rio de Janeiro, RJ, Brazil
e-mail: danielc@cos.ufrj.br

K. A. C. S. Ocaña
e-mail: kary@cos.ufrj.br

M. Mattoso
e-mail: marta@cos.ufrj.br

F. Baião
Federal University of the State
of Rio de Janeiro – UNIRIO,
Rio de Janeiro, RJ, Brazil

F. Baião
e-mail: fernanda.baiao@uniriotec.br

1 Introduction

Cloud computing [1, 2] is a recently proposed and evolving paradigm that provides computing as a service that can be purchased on demand by different types of clients following a pay-per-use model [2]. Differently from Grid [3] and clusters users, cloud users benefit from the concept of elasticity, as they can easily scale resources up and down according to their processing demand, in a virtually unlimited form [1]. Clouds are strongly based on the concept of virtualization in which Virtual Machines (VM) play a fundamental role. This allocation of resources, on demand, provides a new dimension for High Performance Computing (HPC). Although, at a first analysis, clouds were considered not suitable for HPC applications [4, 5], other studies, such as He et al. [6] provide concrete evidences that new developments in cloud infrastructure are significantly improving the scalability and performance of cloud-based HPC systems.

Due to its elastic capability, cloud environments are beginning to be adopted as a platform for running high-performance and distributed applications, such as, large-scale scientific experiments. Clouds, however, have mostly been used for business applications, thus there is a gap between the cloud environment and the management of scientific experiments, especially ones modelled as data-intensive scientific workflows. This new scenario presents several open issues, three of them discussed as follows.

Scientific experiments are based on complex computer simulations that consume and produce large datasets and allocate huge amounts of computational resources [7]. To help scientists in managing resources involved in large-scale scientific simulations, scientific workflows are gaining much interest [7–9]. A scientific workflow is an abstraction that structures the steps of a scientific experiment as a graph of activities, in which nodes correspond to data processing activities and edges represent the dataflow between them [8]. Scientific Workflow Management Systems (SWfMS) allow for defining, executing and monitoring workflow execution. Additionally, a scientific experiment is only considered "scientific" if it can be reproducible. To obtain the same results, scientists have to analyze data from previous executions. This data is called provenance [10, 11].

As the complexity of the scientific workflow grows (e.g. exploration of thousands of parameters and in many cases using repetition structures over dozens of complex activities), running these workflows demands parallelism and powerful HPC capabilities that can be provided by existing clouds. There are several scientific workflows that were designed to be executed in parallel (exploring parameter sweep [12] or data fragmentation [13]) that demand HPC capabilities, such as Montage, a workflow for astronomical analysis [14], EdgeCFD, a computational fluid dynamics workflow [15], X-Ray crystallography analysis [16], orthologous detection [17], phylogenomic analysis [18], evolutionary analysis [19] and phylogenetic analysis [20]. Each one of these workflow executions is composed by hundreds or thousands of parallel tasks that consume several hundreds of files and produce thousands of other files. For each execution in the cloud, scientists should inform some restrictions such as the deadline (the maximum time to wait for workflow completion) and the limit budget (the maximum amount of money to be paid). According to this information the workflow tasks have to be scheduled and executed in parallel on the VMs of the cloud.

Task scheduling is a well-known NP-complete problem even in simple scenarios [21–24]. Therefore, many heuristics have been proposed to address the scheduling problem in different platforms (clusters, Grids). In the last few years many static heuristics have been proposed [25–27] for clusters and Grids. These heuristics try to generate an optimal (or sub-optimal) scheduling plan, i.e. to allocate activities of a workflow onto a set of available machines prior to the workflow execution.

However, scheduling and executing parallel scientific workflows in clouds is still a challenge due to several reasons. Firstly, cloud environments have a unique pricing model that have to be considered to fit in the budget informed by scientists. In clusters and Grids, there is an initial investment (sometimes huge) and a small operational cost paid over time (to pay for energy consumption or maintenance team, for instance). On the other hand, cloud providers allow for acquiring resources on demand, based on pay-per-use pricing

model, e.g. users pay per quantum of time used (e.g. minutes, hours, days, etc.).

Secondly, clouds are changing environments and they may be susceptible to performance variations during the execution course of the workflow, thus requiring adaptive scheduling solutions. Elastic scaling of resources (hardware and software capabilities) is a key characteristic of clouds. To provide this feature, resources are allocated and reallocated as needed by the provider and the users are not aware of those changes. For example, if scientists execute their workflows using Amazon EC2's spot instances [28], VMs can be unallocated or moved as the provider needs more CPU capacity. In the case of Amazon, the demand varies during the years, which means that they may need more resources to handle their Christmas rush than they do the rest of the year, for example. These moves and instabilities can produce negative impacts on the scheduling of workflow tasks.

Thirdly, in clouds, VM failures are no longer an exception but rather a characteristic of these environments. Although the cloud provider tries to guarantee the reliability of the environment, the workflow scheduling has to consider reliability issues when distributing parallel tasks to execute on several distributed VMs.

In this paper, we present an approach based on a heuristic for adaptive scheduling of parallel scientific workflows in cloud environments. This approach is based on a 3-objective weighted cost model that considers execution time, reliability and financial cost simultaneously. This cost model was based on the ideas of Boeres et al. [25]. By using this cost model, an adaptive greedy scheduling algorithm schedules workflows tasks into the many VMs and a load balancing algorithm instantiates and destroys VMs to meet the deadline and to fit in the budget informed by scientists.

The proposed cost model and the greedy scheduling algorithm are highly based on the provenance data of experiments captured and queried at runtime by the cloud workflow engine. Besides providing information that allows for reproducing experiments, the provenance repository also provides the necessary information from previous executions to estimate the execution time of tasks, to analyze volume of data produced by a task or to discover which tasks share input files. This type of information is fundamental for the proposed cost model and scheduling algorithm and allows for a fine tuning.

In order to be evaluated, this approach is implemented within SciCumulus[1] [29, 30]. SciCumulus is a cloud workflow engine that supports parallel execution of scientific workflows in clouds with coupled provenance support. Differently from the current mainstream, SciCumulus automatically adapts the workflow execution according to the current state of the environment, thus benefiting from elasticity. It automatically creates VMs and configures virtual clusters based on these VMs following restrictions defined by scientists. In addition, SciCumulus generates runtime provenance data, i.e. registers information about the workflow being executed and makes this data available for queries at runtime. This allows for the proposed approach to adapt more efficiently. In this way, the main contributions of this paper are:

1. A 3-objective weighted cost model for scheduling scientific workflows on the cloud considering total execution time, financial cost and reliability.
2. An adaptive greedy scheduling algorithm to explore the space of alternative schedules considering changes in the cloud environment and the proposed cost model.
3. A thorough experimental evaluation based on the implementation of the proposed approach in SciCumulus cloud workflow engine. We use SciPhy [20], a real scientific workflow for phylogenetic analysis on a 128-core virtual cluster in Amazon EC2 environment that shows the performance benefits of the proposed approach.

The paper is organized as follows. Section 2 brings related work. Section 3 discusses the formalism for scientific workflows used in this paper. Section 4 introduces the 3-objective cost model that takes into account execution time, financial cost and reliability. Section 5 presents the adaptive greedy scheduling algorithm. Section 6 details SciCumulus architecture used in our experiments, Section 7

[1] http://scicumulus.sourceforge.net/

presents the bioinformatics scientific workflow used as case study. In Section 8, results of the experimental evaluation are analyzed; comparing the proposed approach with existing solutions and finally, Section 9 concludes the paper and points out some future work.

2 Related Work

The adaptive approach based on a cost model presented in this paper was strongly inspired by well-established adaptive query processing techniques [31–33]. In adaptive query processing the focus is on using runtime feedback from database systems (DBMS) to modify query processing in a way that provides better response time and more efficient CPU utilization. In the context of this paper, the query is equivalent to the execution of a task that consumes some data and a combination of parameter values. Since we did not find scheduling algorithms that considered cloud elasticity issues and scientific workflows computations we have structured the related work section in two parts. In the first one we discuss existing scheduling algorithms based on multi-objective cost models, not necessarily for clouds. In the second part we present existing approaches for executing scientific workflows in parallel on the cloud.

2.1 Multi-objective Scheduling

In the literature, there are some works that focus on multi-objective scheduling algorithms, more specifically in bi-objective scheduling algorithms. In general, bi-objective scheduling considers execution time and reliability of the system. However, no work was found that considers financial costs. Assayad et al. [27] introduce a bi-objective scheduling heuristic for DAG of activities according to: the minimization of the schedule length (execution time), and second the maximization of the system reliability. It also uses the active replication of activities to improve reliability. Since they focus on clusters, they do not take into account the financial cost to be minimized.

Qin et al. [26] present a dynamic scheduling heuristic for parallel real-time jobs on heterogeneous clusters. It is assumed that jobs are modeled by DAGs and that they arrive at the system following a Poisson process [34]. The algorithm considers the reliability measure and there is an admission control so that a parallel real-time job whose deadline cannot be guaranteed is rejected. Although this approach focuses on meeting deadlines (restriction) it does not take into account the financial cost or limit budget informed by scientists as well.

Boeres et al. [25] introduce a cost function developed for the Makespan and Reliability Cost Driven (MRCD) algorithm that integrates reliability and performance (execution time) objectives simultaneously in the same formula. This article was the inspiration for the cost model proposed in our approach. However, Boeres et al. do not consider financial costs involved in the execution and the variation of the resources since it was designed to execute in "static" environments such as computing clusters. This cost function was chosen to be extended in this paper because it allows for fine tuning the criteria based on parameter weights, which was not found in other related work.

2.2 Approaches for Executing Workflows in Clouds

The MapReduce model [35] recently gained attention as a model for parallel programming. A range of applications can be parallelized by using this programming model. In several applications, embarrassingly parallel [36] can greatly benefit from this model. However, MapReduce implementations are not focused on parallelizing scientific workflow activities. Thus, in workflows, MapReduce frameworks have to be coupled to existing SWfMS. There are several SWfMS that show coupled MapReduce implementations to their workflow engines such as Kepler + Hadoop [37], MapReduce VisTrails [38] and View [39]. Each one of these SWfMS provides components to implement Map and Reduce functions in its architecture. Despite being presented as a contribution in the articles, the use of cloud is not clear in the proposals. In all approaches the components to be connected to the cloud are not native of SGWfC. In the case of Kepler, the experiments were performed in a cluster using the word count

activity as a case study [37], which does not necessarily reflect the scenarios found in scientific workflows.

One of the existing approaches is Nimrod/K [40]. It proposes mechanisms to provide Many Task Computing (MTC) capabilities to scientific applications. Nimrod/K has a set of components and a run time machine for SWfMS Kepler [41]. Nimrod/K is based on an architecture that uses tagged dataflow concepts for parallel machines. These concepts are implemented as a Kepler director that orchestrates the execution on clusters, Grids and clouds using MTC. Nimrod/K focused on parameter exploration, and differently from SciCumulus it is dependent of Kepler and relies on the pre-existence of a configured cloud environment, thus it does not configure and instantiate the VMs on demand. Different from SciCumulus, Nimrod/K does not adapt the execution of the parameter sweep according to the changes of the environment and to meet deadline and budget constraints.

Hoffa et al. [42] propose the use of VM clusters with SWfMS Pegasus [43] to evaluate the tradeoffs between running scientific workflows in a local environment and running in a virtualized environment. In this pioneer article, their perspective was not to analyze detailed performance metrics, but show that domain scientists can use different environments to perform their experiments. Although they coupled a virtualized environment to a SWfMS, the approach was specific for Pegasus. In addition, the experiments presented by Hoffa et al. do not concern about adaptive mechanisms, i.e. the scheduling of new activities is based on the status of the cloud environment in the beginning of the workflow execution. The lack of a detailed study and performance and scalability metrics of running parallel scientific workflows in cloud environments prevents the reuse of the approach.

Warneke and Cao [44] propose Nephele, a data processing framework that explicitly exploits the dynamic resource allocation offered by clouds. Nephele schedules tasks of a specific job to different types of VMs and manages their instantiation and termination during the course of the execution of a specific job. However, Nephele is disconnected from the concept of scientific workflows and it is focused on parallel data processing, thus not providing parameter sweep features. In addition, the adaptive algorithm of Nephele only considers the number of VMs involved in the execution, and it does not try to adapt the size of the task to be executed either to meet deadlines or to fit into specified budgets.

There is some work in the literature that deals with adaptive scheduling algorithms for cloud environments even when they are not directly related to scientific workflows. Nuage, proposed by Lee et al. [45] is one of these approaches and it uses evolutionary game theory to provide an adaptive and stable application deployment in cloud environments. Differently from SciCumulus, Nuage is focused on the deployment of business applications and it tries to adapt the volume of requests of each application according to the available VMs to use. However, the applications deployed using Nuage did not connect to the concept of scientific workflows and scientific experiments.

3 Scientific Workflow Formalism

The scientific workflow formalism presented in this paper was inspired on Ogasawara et al. [46]. In this paper, a scientific workflow is defined by a Directed Acyclic Graph (DAG) named $W(A,Dep)$. Nodes (A) in W correspond to all workflow activities to be executed in parallel and the edges (Dep) are associated to data dependency between activities in A. This way, $A = \{a_1, a_2, \ldots, a_n\}$.

Given $a_i | (1 \leq i \leq n)$, let $I = \{i_1, i_2, \ldots, i_m\}$ be the input dataset for activity a_i, then $Input(a_i) \supset I$. Also, let O be the output data produced by a_i, then $Output(a_i) \supset O$. 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 $dep\left(a_i, a_j, ds\right) \leftrightarrow \exists O_k \in Input\left(a_j\right) \middle| \, O_k \in Output\left(a_i\right)$ and ds is the size of data transferred between these two activities. Given a workflow W, a set $Ca = \{ca_1, ca_2, \ldots, ca_k\}$ of cloud activities [16] (a synonym of task in the context of SciCumulus workflow engine) 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 $DepCa(ca_i, ca_j, ds) \leftrightarrow \exists r \in Input(ca_j) \,|\, r \in Output(ca_i) \wedge dep(Act(ca_i), Act(ca_j), ds)$. A specific cloud activity ca_i is modeled as $ca(time)$ where $time$ is the total execution time of ca.

Data parallelism in scientific workflows is characterized by the simultaneous execution of various cloud activities of one activity a_i of the workflow W, where each ca_i processes a disjoint subset of the entire input data, which can be represented by a fragment set $F = \{fi_1, fi_2, \ldots, fi_k\}$. The set of fragmented output data fo_k generated at each $k-$th execution have to be merged (following a specific merging criteria) in order to produce the final result (O_i).

Parameter sweep parallelism, which is presented in our experimental evaluation, is defined as simultaneous executions of cloud activities, where each ca_i consumes a specific subset named Pv_i which is the combination of the set of parameters' domain Pt of an activity a_i to be processed. It can be achieved by generating several cloud activities of a specific activity and each ca_i processes a specific subset of the possible parameters values Pv_i. For example, suppose that two cloud activities ca_i and ca_l have parameters pt_1 and pt_2. Suppose also that the domain of possible values for parameters pt_1 and pt_2 are $Dpt_1 = \{x, x'\}$ and $Dpt_2 = \{y, y'\}$. Let us consider that ca_i consumes values x and y for parameters pt_1 and pt_2 respectively, and ca_l consumes values x' and y'. This way, we can define the combinations $Pv_1 = \{x, y\}$ and $Pv_2 = \{x', y'\}$. Consequently, the parameter sweep parallelism can be achieved by processing Pv_1 and Pv_2 in parallel. The several Pv_i are formed by taking the parameter values from the domain in a given order. Thus, Pv_1 is formed by Dpt_1 [1] and Dpt_2 [1]. Pv_2 is formed by Dpt_1 [2] and Dpt_2 [2], and successively. The set of output data O_k generated by the k-th execution needs to be aggregated, since they refer to different executions of the same experiment. In this type of parallelism, $I$is the same for each ca_i.

4 The Proposed 3-Objective Cost Model

A cost model consists of a set of formulas that estimates the costs involved of performing a task in a particular computing environment [47], which in our case is the cloud environment. A cost model is usually implemented in a system that performs several calculations to choose a specific solution. In the context of this paper, this system is SciCumulus cloud workflow engine (detailed in Section 6) that needs the proposed cost model to schedule several cloud activities on a changing set of VMs. The proposed cost model was based on the ideas of Boeres et al. [25].

Let us consider $VM = \{VM_0, VM_1,..,VM_{m-1}\}$ as the set of m virtual machines that composes a virtual cluster and that are available for general use, and each VM_j is associated to a specific computational slowdown index (csi) [25]. The csi value is used in the model since cloud environments typically classify their VM according to processing power. To obtain this information we may query the provenance repository (that is detailed in Section 6) or we can execute a toy program (it can be a simple program to multiply two numbers, or to merge two different input files) to estimate the performance of the VMs. There may be different VMs with different hardware configurations, which produce different performance information related to the toy program. This metric is calculated to be inversely proportional to the computational power of VM_j. Then, a specific virtual machine in VM is modeled as *vm(network, csi, memory, maxGranfactor, partGranFactor, avgTime, prevExecTime)* where *network* is related to the network bandwidth of this VM, *csi* is the computational slowdown index, *memory* is the RAM capacity of this VM, *maxGranFactor*, *partGranFactor*, *avgTime* and *prevExecTime* are information about the previous executions of this VM and they are further explained as follows. The computational slowdown index of a VM can be also represented as $csi(VM_j)$. Let $P(ca_i,VM_j)$be the expected execution time of a cloud activity ca_i in VM_j in the virtual cluster, this way $P(ca_i,VM_j) = ca_i.time \times csi(VM_j)$.

Let us also consider $\varphi(W,VM)$ as a scheduling of all cloud activities of CA of W on VM. Formally, given a workflow W that includes a set

of activities $A = \{a_1, a_2 \ldots, a_n\}$ and a set $CA = \{ca_1, \ldots, ca_k\}$ of cloud activities created for parallel workflow execution, let $\varphi(W,VM) = \{sched_1, sched_2, \ldots, sched_k\}$ for CA. Let us also consider as $sched(ca_i, VM_j, start, end)$ where *start* and *end* are the start and end time of a cloud activity ca_i in VM_j. Since the cloud environment is a changing environment (VMs are instantiated and destroyed during the course of the workflow) we cannot create an a priori scheduling plan. This way, the several *sched* have to be generated during the course of the experiment. We define $ord(sched_i)$ to be the position of $sched_i$ in the sequence of all schedules. We say that $sched_i < sched_j \leftrightarrow ord(sched_i) < ord(sched_j)$.

Since SciCumulus creates a virtual cluster to execute scientific workflows in parallel, let us consider $|VM|$ the number of VMs in the virtual cluster. Then we may represent the cloud activity processing rate (i.e., the number of cloud activities processed *per* unit of time) of a virtual cluster as (upper bound):

$$\beta = \frac{|VM| \times |CA|}{\sum_{i=0}^{k} P(ca_i, VM_j)} \tag{1}$$

Equation 1 presents the ratio between all possible execution combinations of cloud activities and VMs and the total time needed to execute these cloud activities. Since (1) represents the cloud activity processing rate in a virtual cluster, let us define σ as the number of cloud activities in a wait queue. Then we may define the makespan of all k cloud activities in a scientific workflow by [25]:

$$MS = \frac{\sigma}{\beta} = \frac{\sum_{i=0}^{k} P(cai_i, VM_j)}{|VM| \times |CA|} \tag{2}$$

Before scheduling a scientific workflow in the cloud using this model we need to estimate the value of $P(ca_i, VM_j)$ before we can estimate the makespan MS. We are able to use the average value of $P(ca_i, VM_j)$ in all previous executions of workflows by querying this information on the provenance repository of SciCumulus or using an off-the-model estimation. This off-the-model estimation has to be used when there is no provenance data available. When we have available data of previous executions we can analyze the average execution time and use this information as input to the proposed cost model. Note that all information used in this cost model is available at the proposed provenance schema (as detailed in Section 6).

VMs present performance fluctuations: they may fail and may be created or destroyed during workflow execution. In this model we assume that VMs can fail following a Poisson distribution $F(VM_j) \forall\, VM_j \in VM$ with constant value. $F(VM_j)$ expresses the probability of a given number of failures in VM_j occurring in a fixed interval of time (during workflow execution) independently of the time since the last event (last *VM* failure). This information can also be gathered from the provenance repository if the VMs are the same of previous workflow executions. The reliability cost $R(ca_i,VM_j)$ is defined as:

$$R(ca_i, VM_j) = F(VM_j) \times P(ca_i, VM_j) \tag{3}$$

It has to be minimized so the reliability gets near 1. This way, the total reliability is defined by the sum of the reliability of all cloud activities involved in the execution. Considering $\gamma(VM_j)$ as the set of cloud activities already executed in VM_j, the reliability cost associated to VM_j is:

$$R_p(VM_j) = \sum_{\forall X \in \gamma(VM_j)} RC(ca_i, VM_j) \tag{4}$$

Another important subject to be considered in our cost model is the network cost. This cost is mainly associated to data transferring from and to the cloud. Since we are in the SciCumulus context, let us assume that the workflow W reads and writes information to a shared area [16], this way all possible schedules in $\varphi(W,VM)$ read and write the same amount of data from/to the same shared area. So the cost of reading and writing is always the same for each schedule.

To model the cost of transferring data let us consider ca_i to be a cloud activity associated to activity a_i of $(Act(ca_i) = a_i)$ that belongs to W defined as $ca_i(time)$ with $sched(ca_i,VM_k)$. From ca_j cloud activity associated to activity $a_j(Act(ca_i) = a_j)$ with $DepCa(ca_i, ca_j, D_{cai \rightarrow caj})$ and $sched(ca_j,VM_w)$. Considering $VM_k \neq VM_w$ we have to insert a new data dependency in W to represent the data transferring between ca_i and the shared area and from shared area to ca_j. This way, we have to replace $depCa(ca_i,$

ca_j, $D_{cai\rightarrow caj}$) by $depCa(ca_i,\ dm_w, D_{cai\rightarrow caj})$ and $depCa(dm_w, ca_j, D_{cai\rightarrow caj})$ and define $sched(dm_w, VM_k)$ and $sched(dm_w, VM_x)$. The execution time is defined by:

$$\mathrm{dm_w.time} = \frac{\mathrm{Dcai} \rightarrow \mathrm{caj}}{\min\left(VM_k.\mathrm{network}, VM_x.\mathrm{network}\right)} \tag{5}$$

This way, the total transfer time (TT) for all cloud activities of W is (assuming that there are $(k+1)$ transfers for k cloud activities):

$$TT = \sum_{i=0}^{k} dmw.time \tag{6}$$

The total monetary cost $M(\varphi)$ depends on the pricing scheme of the cloud provider (such as Amazon EC2 or GoGrid). Most of the existing cloud providers calculate the payment based on quantum of time, i.e. you pay a specific value *per* quantum of time, and which varies is the granularity of the quantum. For example, Amazon EC2 uses one hour quantum while when using GoGrid prepaid plans where we pre-pay an amount and the provider discounts usage and overage rates per minute. Due to this reason in this model we were inspired in the ideas of Kllapi et al. [48], our total time as a sum of several quanta of time Qt which generated the set $\delta = \{\delta_1, \delta_2, \ldots, \delta_x)$ 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.

$$\omega\left(\delta_i, VM_j\right) \begin{cases} 1, \textit{ if } VM_j \textit{ is running a cloud activity in } \delta_i \\ 0, \textit{ elsewhere} \end{cases}$$

We have also to consider the time that the VMs need to transfer data. This way, our monetary cost is separated in two main parts. The first one is related to the cloud activities execution cost and the second one is related to data transfer cost. Thus, the total execution monetary cost is defined by the following equation:

$$M(\varphi) = \left(Vq \times \sum_{j=1}^{|VM|} \sum_{j=1}^{MS/Qt} \omega\left(\delta_i, VM_j\right)\right) + \left(Vq \times \sum_{i=0}^{k} dmi.time\right) \tag{7}$$

The transfer financial cost $Tr(\varphi)$ also depends on the pricing scheme of the cloud provider (such as Amazon EC2 or GoGrid). Let us consider Vt as the value the scientist has to pay for each unit of transferred data. This way, the total amount to be paid for data transferring is:

$$Tr(\varphi) = Vt \times \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} depT.ds\left(ca_i, ca_j\right) \tag{8}$$

The proposed cost model is then based on execution time, financial cost and reliability criteria. To schedule based on these three objectives we use a weighted cost model where the scientist has to inform the weight of each criterion (it varies from scientist to scientist or from experiment to experiment). This way, for each VM_j in VM that is idle and request for a cloud activity, it is then performed a search for the best ca_i in the list of available cloud activities Ca' (the ones ready to be executed) to execute in VM_j following the 3-objective cost model (9). Given $VM_j \in VM$ as the next VM to execute a cloud activity, we have to find $ca_i \in Ca'$ which minimizes the following cost function:

$$\begin{aligned} & f\left(ca_i, VM_j\right) \\ & = \alpha_1 \times \left(P\left(ca_i, VM_j\right) + \sum_{j=0}^{k} \frac{depT.ds\,(cai, caj)}{VM_j.network}\right) \\ & + \alpha_2 \times R\left(ca_i, VM_j\right) + \alpha_3 \\ & \times \left(Vh \times \left[P\left(ca_i, VM_j\right) + \sum_{j=0}^{k} \frac{depT.ds\,(cai, caj)}{VM_j.network}\right]\right. \\ & \left. + Vt \times \sum_{j=0}^{k} depT.ds\left(ca_i, ca_j\right)\right) \end{aligned} \tag{9}$$

Equation 9 is composed by aforementioned formulas for calculating execution time, reliability cost and monetary cost. Note that the chosen ca_i is the one that satisfies $F(ca_i, VM_j) = min\ \forall\ VM_j \in VM\{f(ca_i,\ VM_j)\}$. The weight associated to each criterion is a variable in the form $0 \leq \alpha_i \leq 1$ which represents the level of relevance for each criterion for the scientist ($i = 1, 2, 3$). By allowing scientists to inform α_i, we provide a way to perform a fine tuning in the workflow execution parameters. The search space for this cost model is a 3-objective one since we try to find the best set of available cloud activities to execute for a specific VM following execution

time, reliability and financial cost criteria. This cost model extends the one proposed by Boeres et al. [25] by adding a new financial dimension that is fundamental when executing workflows in clouds.

This cost model has to be part of a scheduling algorithm and implemented in a computational system. In the following sections we present the adaptive greedy scheduling algorithm proposed in this paper and the cloud workflow engine where both cost model and the greedy scheduling algorithms are implemented.

5 The Adaptive Scheduling Algorithm

In clouds, the search space is three dimensional (execution time, reliability and financial cost). In fact, the scheduling solutions represent the trade-offs between these three criteria. In this paper we focus on solving four different scenarios: (S1) to find the fastest scheduling plan; (S2) to find the plan with the less financial cost; (S3) to find the most secure scheduling plan where cloud activities present minimum fail and (S4) to find a balanced scheduling plan (each one of the criterion—total execution time, reliability and financial cost—have equal weight). Actually, to implement each one of these scenarios we have to set different values of α_i ($0 \leq \alpha_i \leq 1$) in order to focus on each specific criterion.

However, independently of the particular scenario that is being solved we execute the same scheduling algorithm, just varying α_i in the cost model. The proposed adaptive approach is based on a greedy scheduling algorithm [49] and a load balancing algorithm that scales up and down VMs involved in the execution to meet deadline and the limit budget informed by scientists. This adaptive approach is composed by four algorithms: (i) a greedy scheduling algorithm; (ii) a cloud activity grouping algorithm; (iii) a cloud activity perform algorithm and (iv) a load balancing algorithm. We explain each one of those algorithms following in this section.

Algorithm 1 is responsible for choosing the most suitable cloud activity to execute for a given idle VM based on the proposed cost model. Actually, this algorithm chooses the best group of cloud activities (that are encapsulated into a new cloud activity as presented in Algorithm 2) to execute in a specific VM. Algorithm 1 starts by loading the list of available cloud activities without dependency, i.e. ready to be executed, (line 3) and the list of available VMs (line 4). After that, the algorithm initializes the VMs with fundamental information to generate groups of cloud activities (lines 5–10).

Algorithm 1 Greedy Scheduling Algorithm

Input:
W: The scientific workflow
VM: The set of VMs to use (dynamic)
Deadline: The deadline informed by scientists
Budget: the budget informed by scientists
Output:
$\varphi(W,VM)$: The schedule of W on VM

```
1: loadBalance(VM, Deadline, Budget)
2: φ(W,VM)←∅
3: available ←{ca_i ∈ CA| ∀ca_j ∈ CA ¬DepT(ca_i,ca_j, ds)}
4: ready ←{vm_j ∈ VM | ∀vm_j ∈ VM av(vm_j)}
5:  for each vm in VM do
6:    vm.partGranFactor ← 1
7:    vm. execTime ← ∞
8:    vm.prevExecTime ← ∞
9:    vm.maxGranFactor ← ∞
10: end for each
11: while available ≠ ∅ do
12:  for each vm in ready do /* Order VM according to
capacity */
13:       maxGranFactor  ← getMaxGranFactor (vm)
14:       partGranFactor ← getPartGranFactor (vm)
15:       avgTime ← getAvgTime(vm)
16:       if maxGranFactor ≠ ∞ then
17:         clusters ←
group(available,maxGranFactor,avgTime)
18:       else
19:         clusters ← group(available,partGranFactor,
avgTime)
20:       end if
21:     for each x in clusters do
22:       cost ← f(x, vm)
23:       possible ← possible + { sched(x_i, VM_j, cost)}
24:     end for each
25:    chosen ← min(possible)
26:    φ(W,VM) ← φ(W,VM) + chosen
27:    perform(chosen,Threshold)
28:    ready ← ready − {vm} + {vm_j ∈ VM | av(vm_j)}
29:    available ←{ca_i ∈ CA | ∀ca_j ∈ CA ¬DepT(ca_i,ca_j, ds)}
30:  end for each
31: end while
32: return φ(W,VM)
```

When all VMs are initialized, the algorithm starts analyzing if there are available cloud activities to

be executed (line 11). If there is at least one cloud activity to execute the algorithm searches for idle VMs to execute the available cloud activities (line 12). It is important to highlight that the algorithm orders the available VMs according to VM capacity and the scenario. If the scenario is S1, then the algorithm orders the VMs from the least powerful VM (smallest *csi*) to the most powerful VM (largest *csi*). On the other hand, if the considered scenario is S2 then the algorithm orders the list of available VM from the most powerful one (largest *csi*) to the least powerful one (smallest *csi*). If there is an idle VM, the algorithm analyzes if there is a granularity factor or a partial granularity factor associated to this VM. This factor is used to group available cloud activities to encapsulate in a new one (lines 13–20). Then, based on these new cloud activities (*clusters*) the algorithm calculates the most suitable group to execute in the VM by using the proposed cost model (lines 21–27). After that, the list of available cloud activities and available VMs are updated (lines 28–29). At the end of the algorithm the final schedule plan is provided (line 32).

The main focus of Algorithm 2 is to produce new cloud activities by encapsulating two or more cloud activities into a new one. The focus is to reduce communication and data transfer overhead. Every time a VM sends a message requesting for a new cloud activity we face a communication overhead. If we can group cloud activities that consume (or partially consume) the same input files we could reduce this overhead. By doing this, we avoid data transfers that are costly in cloud environments. For example, let us suppose that cloud activity ca_1 consumes files f_1 and f_2 and cloud activity ca_2 consumes f_1, f_2 and f_3. If we schedule ca_1 to VM_1 and ca_2 to VM_2 we have to transfer 2 files to VM_1 and 3 files to VM_2 while if we encapsulate ca_1 and ca_2 into a new cloud activity ca_3 we have to transfer 3 files to the chosen VM. However, one of the fundamental challenges is how to determine group sizes and which cloud activities have to be grouped.

This way we dynamically tune group size based on existing provenance data (to estimate execution time and to verify which files are consumed and produced by each activity). By analyzing provenance data related to previously executed experiments we are able to estimate the execution time of the new cloud activity (two or more cloud activities encapsulated into a new one). In addition, we aim at maintaining the same average execution time of cloud activities associated to the VM.

Algorithm 2 starts by verifying if the granularity factor is null and in this case the group is impossible to be set (line 2). If the number of available cloud activities to be grouped is inferior to the granularity factor, just one group is formed (line 6). On the other hand, if there are more cloud activities than the value of the granularity factor, the algorithm tries to generate several groups by choosing available cloud activities (that preferentially share some input files to avoid data transferring) to be grouped by maintaining an approximate average execution time (lines 11–27).

Algorithm 2 Cloud Activities Grouping

```
Input:
  Ca: list of available cloud activities
  GranFactor: granularity factor
  AvgTime : Average Execution Time
Output:
  Clusters: the list of generated clusters

1: if GranFactor = 0 then
2:    Clusters ← 0
3:    Return
4: end if
5: if GranFactor > Ca then
6:   Clusters ← Ca
7:   Return
8: end if
9: subSetCount ← 0
10: subSet ← {}
11:  while Ca ≠ ∅ do
12:     while (subSetCount < GranFactor) or (Ca ≠ ∅) do
13:        act ← getAvailable(Ca)
14:      subSet ← subSet + act
15:      predExecTime ← getPredictedTime(SubSet)
16:      predAvgTime ← predExecTime / GranFactor
17:      if predAvgTime ≤ AvgTime then
18:         subSetCount ← subSetCount + 1
19:         Ca ← Ca – subSet
20:      else
21:          subSet ← subSet - act
22:      end if
23:     end while
24: Clusters ← Cluster + subSet
25: subSet ← {}
26: subSetCount ← 0
27 end while
28: return Clusters
```

Algorithm 3 is responsible for setting up the granularity factor for each VM in the system. This information is a prerequisite for Algorithm 2. This algorithm starts by executing the cloud activity (or group of cloud activities) (line 1). After that it calculates the average execution time (line 3) and the average execution time of the previous execution (line 4) on the VM. If the new average time is inferior to the previous average time (line 5) the algorithm increases the granularity factor (line 9). On the other hand if the new average time is superior to the previous average time we set the maximum granularity factor (line 6).

Algorithm 3 Cloud Activity Perform Algorithm

Input:
Chosen: a given schedule
Threshold: the threshold to define the maximum granularity factor
Output:
-

```
1: execTime ← execute(chosen)
2: vm ← getVM(chosen)
3: avgTime ← execTime / vm.partGranFactor
4: prevAvgTime ← vm.prevExecTime / (vm.partGranFactor - 1)
5: if (avgTime ≥ prevAvgTime × Threshold) then
6:    vm.maxGranFactor ← partGranFactor
7:    vm.avgTime ← prevAvgTime
8: else
9:    vm.partGranfactor ← vm.partGranFactor + 1
10   vm.avgTime ← avgTime
11:end if
12:vm.prevExecTime ← vm.execTime
```

Algorithm 4 focuses on allowing the system to adapt the amount of resources (VMs) to fit the deadline and the limit budget informed by scientists. This way, this algorithm allows scientists to quickly scale virtual cluster capacity, both up and down, as your computing requirements (deadline and budget) cannot be meet.

Algorithm 4 starts by verifying if the throughput (cloud activities finished *per* second) has varied (line 6). If not, nothing changes in the set of available VMs. If the throughput has reduced, the algorithm simulates the new makespan (MS') and the new financial cost (MC') (lines 7–8). If the new makespan is larger than the deadline and the financial cost is smaller than the informed budget, the number of VMs is scaled up in order to meet the deadline (lines 9–15). On the other hand, if the deadline is met but the financial cost is over budget, then the algorithm scales down the number of VMs involved in the execution (lines 16–23). This reduction in the number of VMs may cause some interruption in the cloud activities. This way, some cloud activities should be rescheduled.

Algorithm 4 Load Balancing Algorithm

Input:
Deadline: The deadline informed by scientists
Budget: the budget informed by scientists
VM: the list of VMs available to use
CA: the list of available cloud activities
Output:
-

```
1: throughput ← ∞
2: prevThroughput ← ∞
3: while (CA ≠ ∅) do
4: VM' ← VM
5:     throughput ← |VM|×|CA| / Σ_{i=0}^{k} P(ca_i,VM_j)
6:     if throughput ≤ prevThroughput then
7:       MS ←simulatMS(VM,CA)
8:       MC ← simulateMC(VM,CA)
9:       if (MS ≥ Deadline) and (MC ≤ Budget) then
10:          while MS' ≤ Deadline and MC' ≤ Budget do
11:            VM' ← VM' + 1
12:            MS'← simulateMS(VM',CA)
12:            MC' ← simulateMC(VM',CA)
13:          end while
14:          VM ← VM'
15:      end if
16:      if (MS ≤ Deadline) and (MC ≥ Budget) then
17:         while MS' ≤ Deadline and MC' ≤ Budget do
18:            VM' ← VM' - 1
19:            MS'← simulateMS(VM',CA)
20:            MC' ← simulateMC(VM',CA)
21:         end while
22:         VM ← VM'
23:      end if
24:      if (MS ≥ Deadline) and (MC ≥ Budget) then
25:        VM ← {}
26:        CA ← {}
27:      end if
28: end while
```

6 SciCumulus Conceptual Architecture

This section briefly describes the architecture of SciCumulus cloud engine used as the computational infra-structure to perform the parallel execution of scientific workflows in clouds and the

data provenance model used as basis of the cost model. Both cost model and the algorithms proposed in previous sections are implemented in SciCumulus.

6.1 SciCumulus Conceptual Architecture

SciCumulus provides support for two types of parallelism: parameter sweep [12] and data parallelism [50]. SciCumulus is designed to distribute, control and monitor parallel execution of scientific workflow activities (or even entire scientific workflows) started from a SWfMS into a cloud environment, such as Amazon EC2 [28]. It is itself distributed and it is based on four-tier architecture:

i **Client Tier**: Its components are placed in the scientists' workstation. It dispatches workflow activities to be executed in the cloud environment using a local SWfMS such as VisTrails [51],
ii **Distribution Tier**: It generates and manages the execution of cloud activities in one or more VMs instantiated in one or more cloud environments,
iii **Execution Tier**: Its components are placed in the several VMs involved with the parallel execution of the cloud activities. It is responsible for executing programs encapsulated in cloud activities and to store provenance data,
iv **Data Tier**: This tier is responsible for storing input data and provenance data consumed and generated by the parallel execution of the workflow. In addition, this tier has information about the environment characteristics collected by an autonomous agent [52].

SciCumulus provides the minimal computational infra-structure to support workflow parallelism with provenance gathering. SciCumulus architecture is simple and may be deployed in any cloud environment (such as Amazon EC2) and connected to any existing SWfMS, diminishing effort from scientists. Figure 1 presents SciCumulus conceptual architecture and its four main tiers.

The Client Tier is responsible for starting parallel execution of workflow activities in the cloud. The components of the client tier are deployed in an existing SWfMS such as VisTrails. The components installed in the SWfMS are responsible for transferring data to and from the cloud and to start the parallel execution in the cloud. The components of the client tier launch the execution process of cloud activities in the cloud

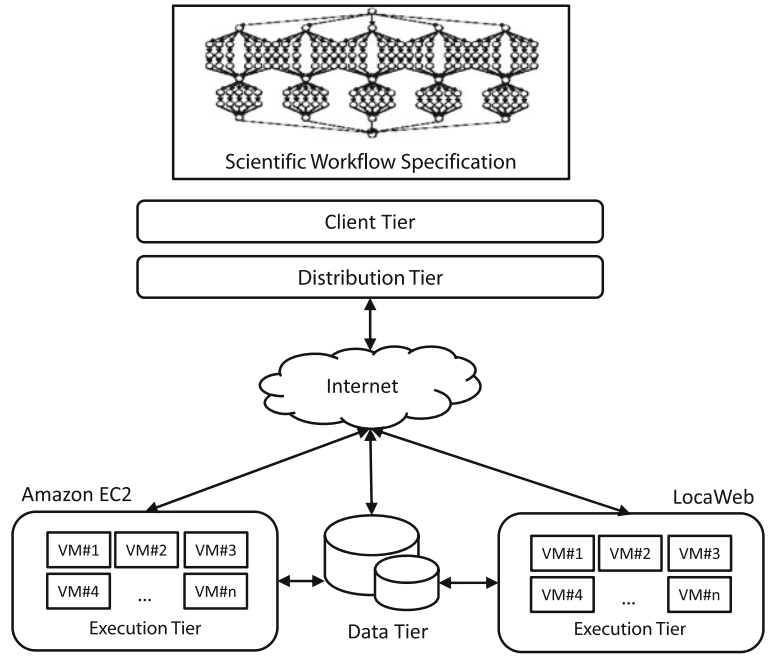

Fig. 1 SciCumulus conceptual architecture

environment and communicate directly to the Distribution Tier.

The Distribution Tier manages the execution of parallel activities in cloud environments by creating and managing cloud activities that contain the program to be executed, its parallel strategy, parameters values and input data to be consumed. Specifically in this tier we have two components that have to be better explained. The first one is the encapsulator. The encapsulator generates all cloud activities to be executed following a specific fragmentation process or a parameter sweep as presented in Section 3. Based on these cloud activities the Scheduler defines which VMs receive a specific cloud activity to execute based on VM request for cloud activities. At this point, the scheduler executes the proposed greedy algorithm to determine the best cloud activity for a specific VM that is idle. Note that the VMs may be provided by any cloud provider. The scheduler has to take into account the available VMs, the permissions to access VMs and the computational power of each one of them (information that is retrieved from the provenance repository of the data tier). The scheduler in SciCumulus may work in two different modes: a static one and an adaptive one. In the static mode, the scheduler considers a fixed amount of available VMs in the beginning of the execution. On the other hand, the adaptive mode aims at analyzing the state of the cloud environment during the course of the execution in order to use more VMs and to adapt the number of cloud activities to be sent to a VM according to the available VMs.

The Execution Tier is responsible for invoking the programs associated to the activities of the workflow in many VMs available for use. The components of the execution tier may be deployed in more than one cloud provider. Finally, the Data Tier contains all repositories of data used by SciCumulus. It has the provenance repository that contains important provenance data [11] collected during the course of the workflow. It also contains information about the types of available VMs, the instantiated VMs at the moment and locality characteristics. This information is captured by an autonomous agent that monitors the environment looking for changes in the environment (new available VMs or destroyed VMs). The data tier also contains all input data consumed by several VMs in the parallel execution. To the best of authors' knowledge, none of the existing cloud environments offer native support to collect provenance and any other means to store provenance metadata produced by scientific experiments. However, there are some papers that highlight the importance of the subject as in Muniswamy Reddy et al. [53], where the authors discuss some alternatives to storage of provenance using cloud computing services offered by Amazon EC2 and using the PASS system [54].

6.2 SciCumulus Provenance Model

SciCumulus is designed and implemented for operating in the cloud as well as its provenance model. It represents all information about the experiment being executed and about the cloud environment itself. This provenance model is based on the proposed Open Provenance Model (OPM) recommendation [55]. The OPM is a recommendation that is open from an inter-operability viewpoint but also with respect to the community of its contributors, reviewers and users. The main idea of OPM is to represent the causal relations between processes, agents, artifacts and roles involved in a parallel workflow execution. The OPM is not directly instantiable in a database, but it is a standard representation of provenance data for most SWfMS [10, 11, 56].

Figure 2 presents the conceptual provenance schema designed for SciCumulus. All information related to previous executions of scientific workflows used in the cost model and the scheduling algorithm is captured from the provenance repository of SciCumulus. This model is represented as a Unified Modeling Language (UML) [57] model and it was modeled based on requirements elicited with scientists. All information of the provenance model is captured by the components in the execution tier of SciCumulus and by the autonomous agent that captures the information about the cloud environment. This provenance model is composed by four main parts: (i) elements that represent the processes executed in the cloud; (ii) elements that represent the artifacts consumed and produced by the workflow execution, (iii) elements that represent the temporality

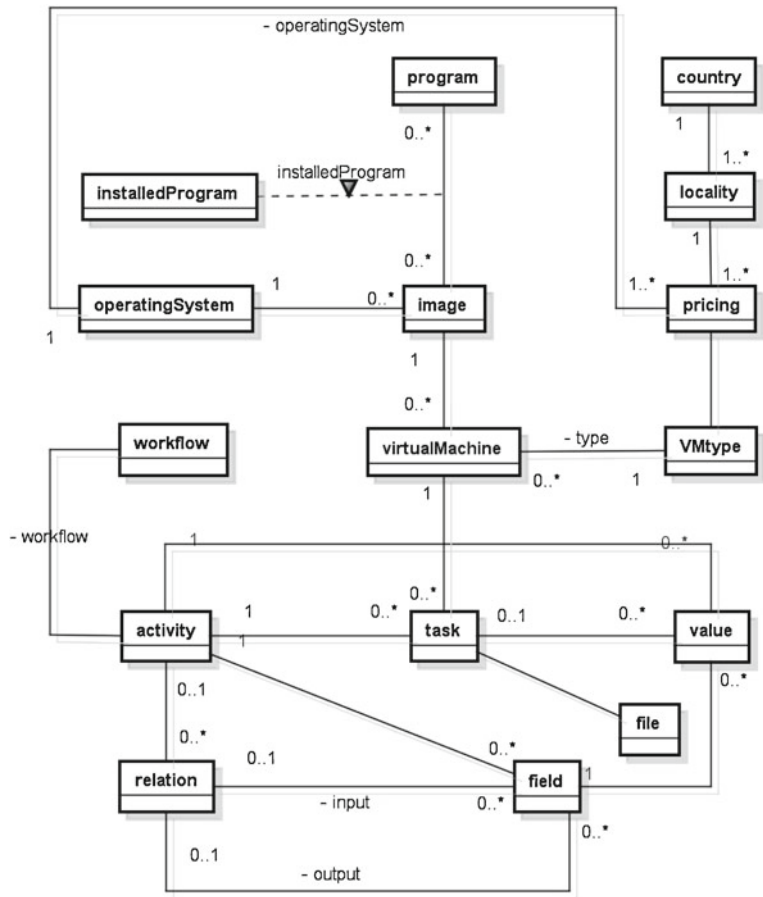

Fig. 2 SciCumulus provenance schema

of data and, (iv) elements that represent information about the cloud environment.

Since this provenance schema is based on the OPM recommendation the classes *InputValue*, *OutputValue*, *VirtualMachine* correspond to the conceptual representation of an artifact since both classes have the same semantics, i.e. model the same digital structures in computational systems (parameters, database, files, and instances).

The classes *Activity* and *Task* are mapped as an OPM process. An OPM process represents one or more actions that consume or produce artifacts. The class *Provider* represents an OPM user. All the provenance data in SciCumulus is generated at runtime, i.e., during workflow execution. This runtime provenance allows SciCumulus to use provenance data for scheduling. In existing approaches such as VisTrails [51], provenance data is only generated at the end of the execution of the workflow.

6.3 SciCumulus Implementation Details

For the experiments executed in this paper, we have deployed SciCumulus architecture on top of Amazon EC2. Deploying a workflow engine like SciCumulus into the Amazon EC2 environment is not a simple task to accomplish. Mainly because of the adaptations that have to be performed on the conceptual architecture due to environment restrictions. SciCumulus current version was developed using Java Version 6.27. The components were implemented based on MPJ [58]. The provenance data is persisted using PostgreSQL relational database version 8.4.6 that was configured in a dedicated Amazon EC2 VM, as well. The client tier components were implemented to connect to the VMs in which the distribution tier components were deployed using Secure Shell (SSH), and send/receive the data files directly to a shared file system in the cloud. Although we use a secure

connection to transfer data (via SSH), security issues are outside the scope of this paper. Security issues in scientific workflows are discussed by Gadelha and Mattoso [59].

SciCumulus uses a shared file system (or any equivalent shared area) to manipulate input and output files. In order to provide this shared file system, we have configured an Amazon Elastic Block Storage (EBS) volume [28]. EBS volumes storages can be accessed by EC2 VMs and their lifetime is independent from the lifetime of the VMs used by SciCumulus execution tier. In other words, this volume is persistent. However, EBS restricts the data size to be stored, and this poses a serious limitation since typically large volumes of scientific data is produced. Following the approach proposed by Jackson et al. [5] the input data is currently put in the local storage of the VM and all output data is stored in Amazon Elastic Store (S3) [28]. To connect the VMs with Amazon S3 we used Subcloud [60]. Subcloud is a shared enterprise file system built on top of Amazon S3. Using Subcloud, we are able to mount a directory on each VM and point to one single bucket in Amazon S3, creating a virtual Shared File System.

In SciCumulus architecture the Distribution tier is responsible for creating a virtual cluster in the cloud environment. Since we are using MPJ, we have to define the head node (which is a VM labeled with rank 0—a common number used to identify a specific process in a set of running processes) and the workers nodes (rank 1, 2 and so on). To setup a virtual cluster we used Amazon EC2 API Tools [28]. A custom image (AMI) for the execution VMs was developed. The Distribution tier queries the provenance repository of types of images to be instantiated in VMs and using the Amazon API starts the necessary number of VMs on demand. It creates a list of IPs of instantiated VMs and this list is used to create the *machines.conf* file that informs the available VMs for MPJ framework to use. The only disadvantage of using this approach is that when a change is needed in any existing image, a new image should be created (a process that usually takes a long time to be performed) and the provenance schema has to be updated in order to use the new image.

In previous work [16, 61] we have evaluated SciCumulus by executing static and simulated dynamic experiments for parameter sweep scenarios, using a simplified scheduling algorithm that does not take the proposed cost model into account. The goal was to make a preliminary analysis of the viability of a parallel execution of a scientific workflow in the cloud, before starting the implementation of an adaptive mechanism such as the one proposed in this paper. The next section presents the case study used to evaluate the proposed cost model and the adaptive scheduling algorithm.

7 Data-intensive Phylogenetic Analysis Workflow

This section presents the workflow used as a case study in this paper. In the last few years, phylogenetic analysis experiments are evolving in a fast pace due to new technologies such as new DNA-sequencing methods and scientific apparatus [62]. Comparative genomics is one of many bioinformatics fields that aim at computationally comparing hundreds of different genomes [63]. Many types of bioinformatics applications associated to this field, such as Multiple Sequence Alignment (MSA), Homologues Detection and Phylogenetic Analysis are increasing in scale and complexity [64]. Managing genomic experiments is far from trivial, since they are computationally intensive and process large amounts of data. As they are based on a pipeline of scientific programs, computational genomics experiments have been assisted by scientific workflows techniques. Especially in phylogenetic analysis workflows, scientists perform a specific set of activities to produce a set of phylogenetic trees, which are used to infer evolutionary relationships between homologous genes represented in the genomes of divergent species.

In order to model a phylogenetic analysis experiment as a scientific workflow it was proposed SciPhy [20]. SciPhy workflow is a parameter sweep one where the same workflow is executed for each file in a given large input dataset. It is composed by five main activities. The first three activities are: multiple sequence alignment (MSA), search for the best evolutionary model, and construction of phylogenetic trees, and they respectively execute the following bioinformatics

applications: MSA programs (MAFFT [65, 66], Kalign [67], ClustalW [68], Muscle [26] and ProbCons [69]), ModelGenerator [70], and RAxML [71].

The last two activities represent the Phylogenomic Analysis: concatenation of MSA to obtain a "superalignment" [72], and construction of phylogenomic trees. The programs that respectively execute these activities are: a Perl script to concatenate each MSA (generated by different MSA programs—one concatenation per MSA program), and RAxML.

The general steps of SciPhy are presented in Fig. 3. The first activity of SciPhy constructs individual MSA using five MSA programs—ClustalW, Kalign, MAFFT, Muscle, and ProbCons—with default parameters. Each MSA program receives a multifasta file as input, then producing a MSA as output. Each MSA is tested at the second activity to find the best evolutionary model using ModelGenerator, and both of them (individual MSA and evolutionary model) are used in the third activity to generate phylogenetic trees using RAxML with 100 bootstrap replicates [62]. Consequently, we obtain several trees for each one of the MSA programs.

In the fourth activity, all individual MSA are used as input for a Perl script, in which they are concatenated thus obtaining a superalignment as output. Since we are exploring five different MSA programs, we can only obtain five superalignments. Each superalignment jointly with the specific pre-chosen evolutionary models (BLOSUM62, CPREV, JTT, WAG, or RtREV) is used as input to construct supermatrix phylogenomic trees.

In phylogenomic analysis activity, there is the option to concatenate multiple gene sequences to construct phylogenetic trees on the genomic level, also known as "genome trees", "supermatrix trees" or also called "supermatrix phylogenomic trees". Trees that have more phylogenetic signals are less susceptible to stochastic errors than those built from a single gene. Nevertheless, phylogenomic trees have held the promise of minimizing anomalies by the sheer power of genome-scale data as they are based on the maximum quantity of genetic information. A phylogenomic tree should be the best reflection of the evolutionary history of the species.

Since we aim at executing a parameter sweep in SciPhy, each one of the first three activities is going to be executed for a different input file containing several sequences (multi-fasta file). Each one of these executions can be performed in parallel. However, the MSA concatenation is a non-parallel cloud activity and just one instance of it is allowed per MSA program. The evolutionary

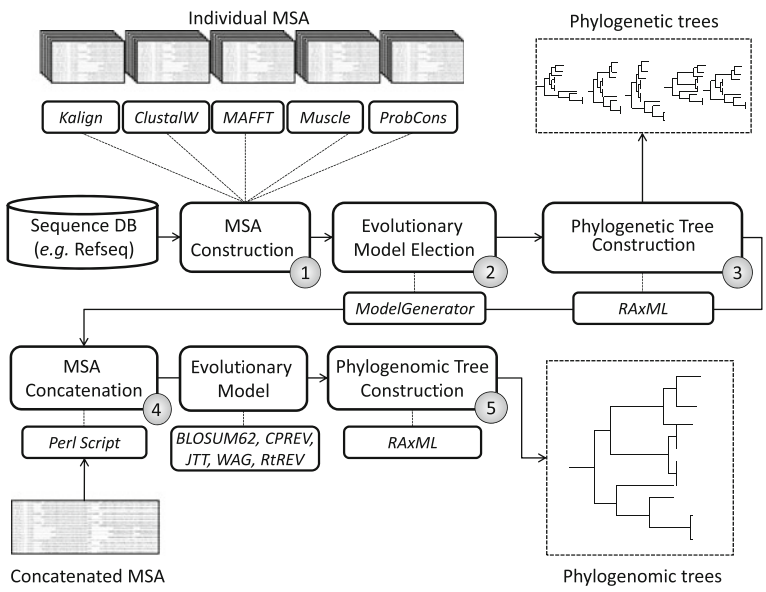

Fig. 3 Conceptual view of SciPhy

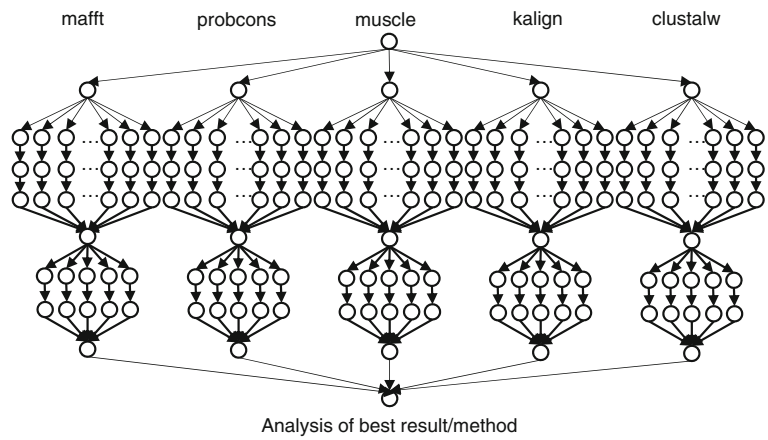

Fig. 4 Parameter sweep for SciPhy

analysis and the construction of the tree activities can be parallelized for each one of the five available methods. A representation of the parallel execution of SciPhy can be viewed in Fig. 4. Each one of the circles represents a different cloud activity to be executed in parallel. Note that the degree of parallelism changes during the execution course of the workflow. For example, if we process 200 multi-fasta files as input. The first part of the workflow (first three activities) is going to have 200 cloud activities for each MSA method. Then, in the second part (last two activities) we are going to have just five cloud activities (one for each chosen method).

8 Experimental Results

This section presents the configurations used to model and execute SciPhy using SciCumulus and experimental results achieved. The central idea of this section is to compare the proposed adaptive approach (cost model and adaptive greedy scheduling algorithm) with existing solutions that can be used to parallelize workflows that need HPC in clouds such as MapReduce [35] implementations (e.g. Hadoop [73]). To evaluate SciCumulus adaptive approach, we have modeled SciPhy workflow as presented in Section 6 on a distributed cloud environment based on CentOS 5.0 operating system machines and we compared the proposed approach with Hadoop. As we discussed at Section 2, we did not find scheduling algorithms that considered cloud elasticity issues and scientific workflow computations. This way, we have chosen to compare the proposed approach with Hadoop implementation, since many of the existing SWfMS parallel approaches are based on Hadoop. In the adaptive evaluation we compared our approach with ideal curves of scheduling and parallel performance measures of speedup, typically the ideal linear speedup curve.

First in this section we briefly describe Hadoop and MapReduce model. Then we describe the environment used for executing the case study and the experiment configuration. After that, we present and discuss about experimental results. Our experimental results are composed by three main parts. In the first one we compare SciCumulus performance with the proposed cost model and scheduling algorithm (without scaling VMs up and down) with Hadoop. In the second part we discuss about results achieved while scaling resources up and down (adaptive approach). In the third part we analyze financial costs of the executions and data transfer.

8.1 MapReduce and Hadoop

MapReduce [74] is a programming model that aims at easing the parallelization of applications. MapReduce deals with large volumes of data to be processed in parallel. Using MapReduce, data can be partitioned into a set of intermediate key-value pairs by a map function, and these intermediate results are merged into the final result using the reduce function following a specific criteria. The main idea behind MapReduce

is to transparently provide for users data partitioning, scheduling, load balancing, and fault tolerance mechanisms. To use MapReduce, two main functions have to be programmed: (i) a map function in the form map(in_key,in_val) → list(out_key,intermediate_val) and (ii) a reduce function in the form reduce(out_key, list(intermediate_val)) → list(out_val). Each one of these functions is domain-specific. In the context of scientific workflows, a new map and a reduce functions should be implemented for each different type of activity. MapReduce was initially used in business applications such as document indexing. In the last few years its use is being extrapolated for scientific domains [75].

One of the most famous and used implementations of MapReduce is Hadoop [76], a framework developed by Apache group that provides the MapReduce core component and the Hadoop Distributed File System (HDFS) natively. In the Hadoop architecture one central process acts as master and coordinates MapReduce tasks (i.e. parallel executions of activities) while all other processes act as workers on different nodes. Workers execute tasks that are generated and distributed by the master node. The same worker is able to execute several maps and reduces tasks at the same time. In addition, implementations of MapReduce such as Hadoop are being used in cloud [1] environments such as Amazon EC2 [28] to ease the access of large amounts of computing power to run data-intensive tasks, including scientific ones.

Although these MapReduce functions jointly with the Hadoop environment variables can be implemented and configured (respectively) by computer specialists, it may be a very complex task to be executed by scientists with little (or none) computational background. In addition, by executing some activities of an experiment using MapReduce decoupled from the SWfMS and without provenance support may produce undesirable results, such as data loss or, more importantly, the inability to reproduce the scientific experiment. These problems may occur as part of the produced data (the one that is produced in the cloud or in a cluster) is not stored in the provenance schema of the SWfMS, for example.

8.2 Environment Setup

For the experiments executed in this article, we have deployed SciCumulus and Hadoop on top of Amazon EC2. Amazon EC2 is one of the most popular cloud computing environments, and many scientific and commercial applications are being deployed on it. Amazon EC2 provides several different types of VMs for scientists to instantiate and use. Each one of them has unique characteristics (CPU power, RAM and storage capacity).

There are several types of VMs, such as micro (EC2 ID: t1.micro—613 MB RAM, 1 core, EBS storage only), large (EC2 ID: m1.large—7.5 GB RAM, 850 GB storage, 2 cores), extra-large (EC2 ID: m1.xlarge—15 GB RAM, 1,690 GB storage, 4 cores), high CPU extra-large instance (EC2 ID: c1.xlarge—7.5 GB RAM, 850 GB storage, 8 cores), and Quadruple Extra Large Instance (EC2 ID: cc1.4xlarge—23 GB RAM, 1,690 GB storage, 8 cores). In the experiments presented in this paper we have considered just Amazon's micro and large type.

Each one of the instances uses quad-core Intel Xeon processors @ 2.33 GHz. Each instantiated VM in the phylogenetic experiments presented in this paper uses Linux Cent OS 5 (64-bit), and it was configured with the necessary software and libraries like MPJ [77] and the bioinformatics applications. All instances were configured to be accessed using SSH without password checking (although this is not recommended due to security issues). Additionally, these images (ami-e4c7368d and ami-ceb949a7) were stored in the cloud as well and SciCumulus creates the virtual cluster to execute the experiment based on these images. The same images were used to execute Hadoop. In terms of software, all instances, no matter its type, executes the same programs and configurations. According to Amazon, all VMs were instantiated in the US East—N. Virginia location and follow the pricing rules of that locality.

8.3 Experiment Setup

To execute SciPhy, we have varied the MSA program and the evolutionary model used in the scientific workflow (see possible variability of MSA

construction and Evolutionary Model Election activities in Fig. 3). Our executions use as input a dataset of multi-fasta files of protein sequences extracted from RefSeq release 48 [78]. This dataset is formed by 1,600 amino acid multi-fasta files and each multi-fasta file is constituted by an average of 10 sequences.

To perform SciPhy, once downloaded, each input multifasta file is aligned to obtain an alignment using the following MSA programs: ClustalW version 2.1, Kalign version 1.04, MAFFT version 6.857, Muscle version 3.8.31, and ProbCons version 1.12. Each alignment is used as input to the program ModelGenerator version 0.85 that produces an evolutionary model as output. Then, both of them, alignment and evolutionary model, are used as input to construct phylogenetic trees using RAxML-7.2.8-ALPHA. We also have varied the MSA program and the evolutionary model used in the scientific workflow (see possible variability of MSA concatenation and Evolutionary Model activities in Fig. 3). All the resultant alignments (from each MSA program), obtained from Phylogenetic Analyses, are concatenated to produce a superalignment. Then, this superalignment is tested with five evolutionary models (BLOSUM62, CPREV, JTT, WAG, and RtREV) and both of them, superalignment and evolutionary models are used as input to construct phylogenomic trees using RAxML-7.2.8-ALPHA. SciPhy workflow was modeled using SciCumulus and Hadoop 0.21.0.

In order to implement each one of these activities in Hadoop we had to implement specific Map and Reduce functions. Since the input of the standalone MSA programs is a single multifasta file (or many multifasta files), meanwhile, in our case, the input format of the MSA activities in Hadoop is a set of multifasta files. We collect those uploaded input files from HDFS to create key-value pairs for the Mapper. The Mapper mainly creates a java process to call the specific program (ClustalW, Kalign, MAFFT, Muscle or ProbCons). The Map key is the filename, and the Map value includes the full HDFS path for each uploaded input files. Each map task downloads the assigned input file from HDFS, and it passes this input to run the associated program. The next activities in the workflow follow the same approach. For each program that is invoked, a new Map and Reduce function had to be implemented (for ModelGenerator, RAxML and for the superlignment script), but these new activities are going to consume the produced data from the previous activity. The reducer is responsible for collecting all intermediate outputs and groups all of them in one single output file to be transferred to the scientist's machine.

8.4 Cloud Activity Execution Time Distribution

Analyzing the provenance repository of SciCumulus we could build the histogram of the execution time for all cloud activities of the SciPhy workflow, as presented in Fig. 5. Based on the information obtained from this repository, it is possible to calculate the average (1,703.5 s) and standard deviation (108.3 s) of each cloud activity. Additionally, scientists can use the provenance repository to check which parameter generated the desired result or submit queries like "what is the average execution time of MAFFT cloud activities that executed in 16 cores?".

The main advantage of having such distribution of execution times is that we can schedule more computing intensive cloud activities to more powerful VMs if we are focusing on performance. On the other hand, we can schedule more computing intensive cloud activities to cheaper VMs if we are focusing on financial cost. Such distribution can benefit from the proposed cost model. In the next subsection we present performance results for the cost model and scheduling algorithms for each one of the scenarios aforementioned.

8.5 Non-adaptive Performance and Scalability Analysis of SciCumulus

In order to perform a fair comparison of SciCumulus scheduling approach with Hadoop we executed SciPhy in SciCumulus without varying the number of VMs involved in the execution (since Hadoop does not scale the number of VMs during execution) and consuming 200 input files (each one of the files contains about 10 sequences to be aligned). In this first part, we execute real

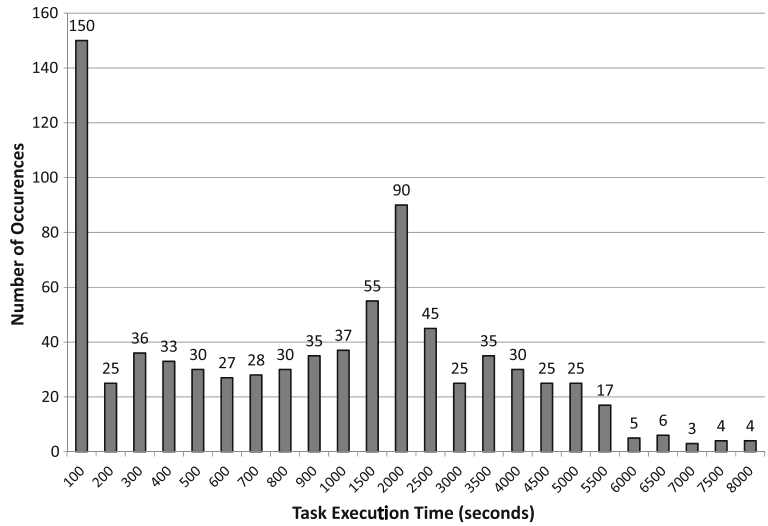

Fig. 5 Cloud activities execution time distribution

parameter explorations using a "static" scenario, where there are no new VMs or VM destructions during the course of the workflow. In this way, we plan to analyze the performance and scalability of SciCumulus scheduling based on the proposed cost model, fixing the number of used VMs in each execution. For each workflow execution to compare SciCumulus with Hadoop, half of the instantiated VMs are large ones and the other half are micro VMs. This way, we try to model a real scenario and to explore the advantages of the proposed cost model in the workflow activity scheduling by varying the type of used resources.

Firstly, we adjusted the cost model for focusing on performance (scenario S1) by setting $\alpha_1 = 0.9$, $\alpha_2 = 0.05$ and $\alpha_3 = 0.05$. The execution time and speedup results are presented in Figs. 6 and 7, respectively, where we can observe that from Figs. 6, and 7 the total execution time of SciPhy workflow using both SciCumulus and Hadoop decreases, as expected, when provided more VMs (and consequently more virtual cores). However SciCumulus outperformed Hadoop when using from 1 to 128 virtual cores.

This is due to two reasons. The first one is that Hadoop presents serious overhead to generate

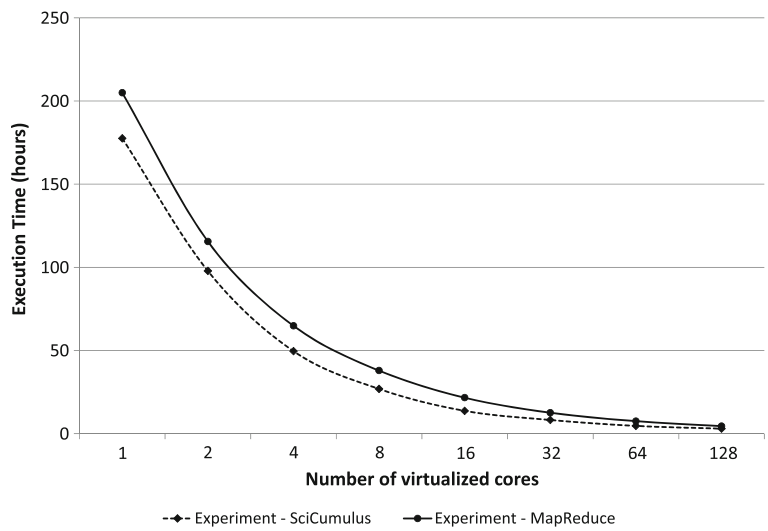

Fig. 6 Execution time for scenario S1

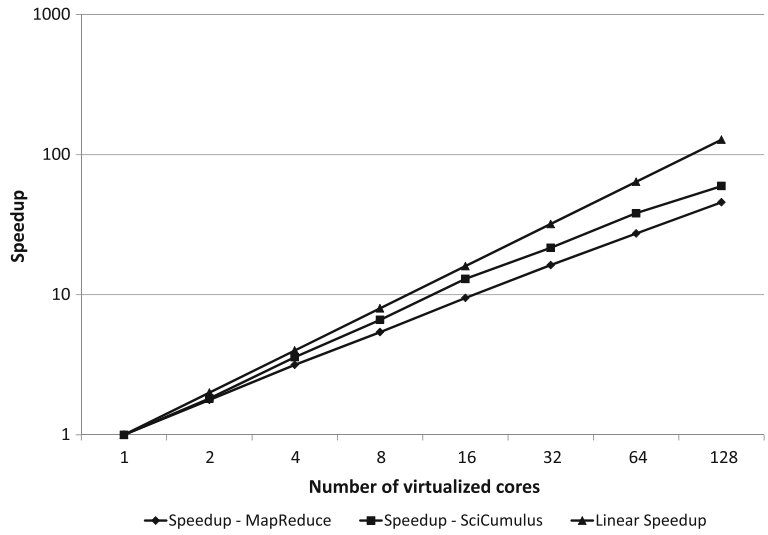

Fig. 7 Speedup for scenario S1

maps. This map generation phase consumes several seconds of each activity execution in the workflow. The second one, SciCumulus scheduling based on the proposed cost model scheduled more intensive cloud activities to large VMs while less intensive cloud activities were scheduled to micro VMs.

This way, since the *csi* of large VMs is greater than the *csi* of micro VMs, the total execution time is reduced. For example, when we execute SciPhy with 64 cores, SciCumulus executed in 4.65 h while Hadoop executed in 7.49 h. It is difference of 37.9 % on the overall performance. This execution led to a speedup of 27.36 (Hadoop considering 64 cores available) and 38.20 (SciCumulus considering 64 cores available). Even when we execute SciPhy with 128 cores the performance gain is considerable. Using 128 cores SciCumulus executed in 2.97 h while Hadoop executed in 4.48 h. It is difference of 33.1 % on the overall performance. This execution led to a speedup of 45.73 (Hadoop considering 128 cores available) and 59.71 (SciCumulus considering 128 cores available). Analyzing the speedup in Fig. 7, we can state that SciCumulus reached a near linear speedup from 2 to 32 cores, and suffered a small degradation when using 64 up to 128 cores. Acquiring more than 64 VMs for execution may not bring benefit or generate a good speedup, particularly if the number of cloud activities becomes closer to the number of VMs.

In clouds there are many factors that can harm speedup. For example, in parallel computers the speedup value is impacted by serial portions of the code and communication between processors, whereas in the cloud, we have to consider other factors such as heterogeneity of the environment, performance fluctuations due to the virtualization and high communication latency [1].

Secondly, we adjusted the cost model for focusing on monetary cost (scenario S2) by setting $\alpha_1 = 0.05$, $\alpha_2 = 0.05$ and $\alpha_3 = 0.9$. The execution time and speedup results are presented in Figs. 8 and 9, respectively. Differently from the execution focused on performance, in this case we have chosen to schedule more intensive cloud activities to micro VMs (which are cheaper) while less intensive cloud activities were scheduled to large VMs (which are more expensive).

However, SciCumulus did not outperform Hadoop when using from 2 to 128 virtual cores, which is acceptable since the focus, is on financial cost instead of performance. SciCumulus only outperformed Hadoop using one single core since there is only a single option to schedule cloud activities. From 2 cores to 128 the performance using SciCumulus was degraded because the least powerful VMs (the ones with the smallest *csi*) were preferentially chosen (since they are cheaper) while in Hadoop the scheduling does not take into account the monetary costs involved in the execution.

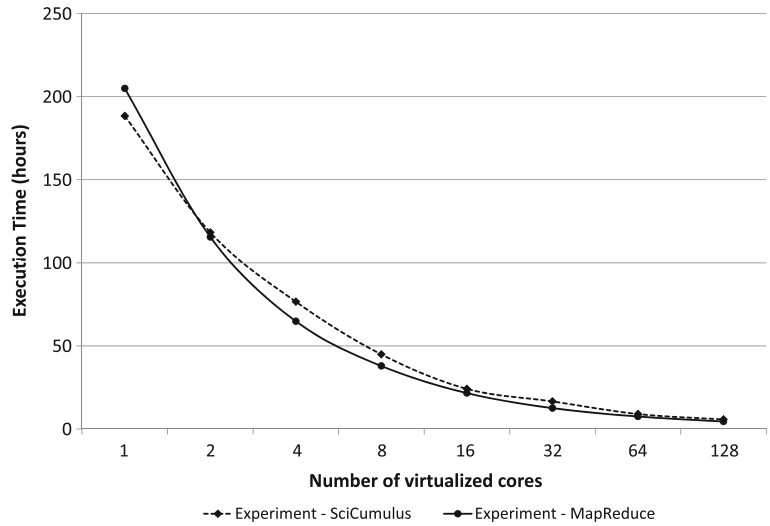

Fig. 8 Execution time for scenario S2

This approach is beneficial because if we are executing the experiment in a provider in which the quantum is 1 min we are going to spend less because intensive cloud activities are executed in cheaper VMs. On the other hand, if the quantum is 1 h, we are not going to spend less financial resources, however, since less intensive cloud activities are scheduled to the powerful VMs, if we are focusing on financial cost, when the load balancing algorithm is executed, it is going to destroy more expensive VMs, thus stopping just less intensive cloud activities (these cloud activities has to be rescheduled).

When we execute SciPhy focusing on financial cost with 64 cores, SciCumulus executed in 8.99 h while Hadoop executed in 7.49 h. It is a "negative" difference of 16.7 % on the overall performance. This execution led to a speedup of 27.36 (Hadoop considering 64 cores available) and 20.94 (SciCumulus considering 64 cores available). Analyzing the speedup in Fig. 9, we can state that Hadoop reached a better speedup than SciCumulus, and both suffered a small degradation when using 64 up to 128 cores. However the difference of speedup is acceptable since the main focus here is on financial cost instead of performance.

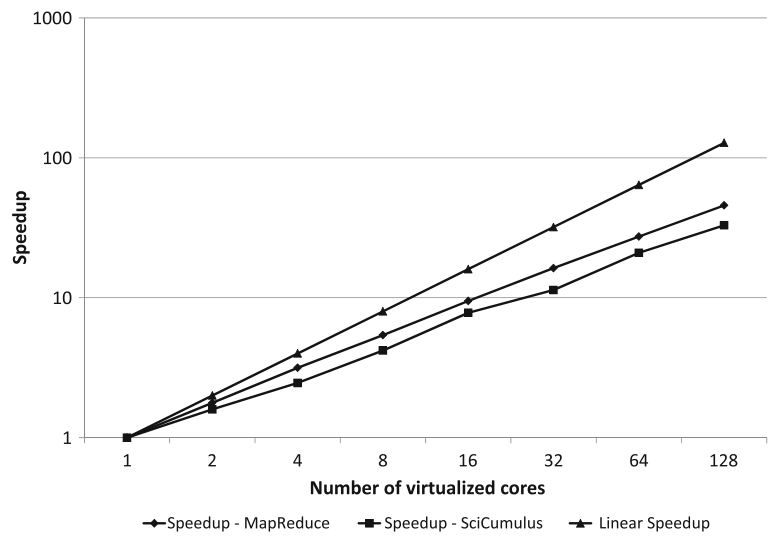

Fig. 9 Speedup scenario S2

Thirdly, we adjusted the cost model for focusing on reliability (scenario S3) by setting $\alpha_1 = 0.05$, $\alpha_2 = 0.9$ and $\alpha_3 = 0.05$. The execution time and speedup results are presented in Figs. 10 and 11, respectively. Since we are focusing on reliability we have to calculate the probability of a given number of failures in the VMs involved in the execution. As we could note by querying provenance data, micro VMs degrade faster than large VMs. Consequently, the failure probability is higher in micro VMs.

This way, the cost model and the scheduling algorithms schedules intensive cloud activities to large VMs while less intensive activities are scheduled to micro VMs. In fact, we avoid re-executing intensive cloud activities because they are scheduled to be executed in more reliable VMs. This way, since large VMs are more reliable than micro VMs, the behavior of this scheduling is similar to the first scenario. Similarly to the first scenario (S1) SciCumulus outperformed Hadoop when using from 1 to 128 virtual cores.

Since the failure probability of large VMs is smaller than the failure probability of micro VMs, the total execution time is reduced. For example, when we execute SciPhy with 64 cores, SciCumulus executed in 5.43 h while Hadoop executed in 7.49 h. It is difference of 34.7 % on the overall performance. This execution led to a speedup of 27.36 (Hadoop considering 64 cores available) and 32.68 (SciCumulus considering 64 cores available). Similarly to the results achieved in scenario S1, there were performance gains up to 128 cores (approximately 1 h difference). Analyzing the speedup in Fig. 11, we can state that SciCumulus reached a near linear speedup from 2 to 16 cores, and suffered a small degradation when using 32 up to 128 cores.

In the last scenario, we adjusted the cost model for providing equal focus for all criteria (scenario S4) by setting $\alpha_1 = 0.33$, $\alpha_2 = 0.33$ and $\alpha_3 = 0.33$. The execution time and speedup results are presented in Figs. 12 and 13, respectively. Since we have attributed equal weights for each criterion, performance reached was very similar to Hadoop as can be stated in Fig. 12. The main difference in performance was due to the overhead imposed by Hadoop to generate maps and reduces.

When we execute SciPhy with 64 cores and equally weighted parameters for the cost model, SciCumulus executed in 5.91 h while Hadoop executed in 7.49 h. It is difference of 21.02 % on the overall performance. This execution led to a speedup of 27.36 (Hadoop considering 64 cores available) and 31.92 (SciCumulus considering 64 cores available). Both SciCumulus and Hadoop presented very similar results in this scenario (S4).

In order to analyze the overhead imposed by SciCumulus when processing large volumes of data, we executed SciPhy, using 16 cores, varying the number of input files and the scenario (weights associated to α_i in the cost model). This

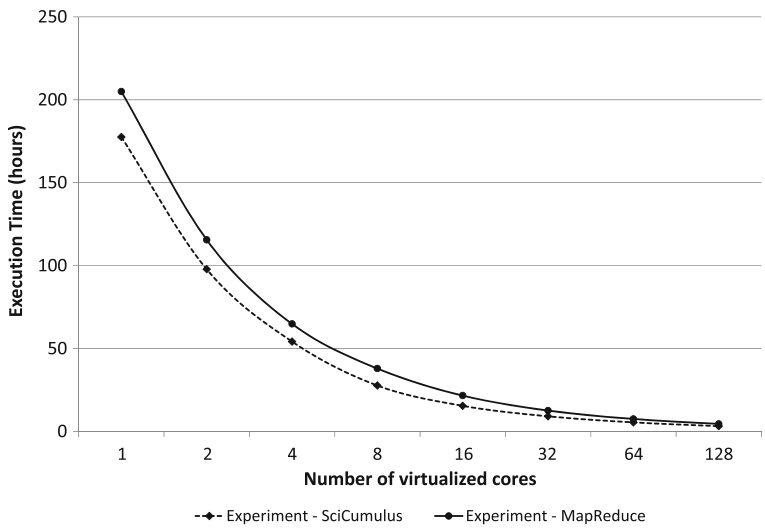

Fig. 10 Execution time for scenario S3

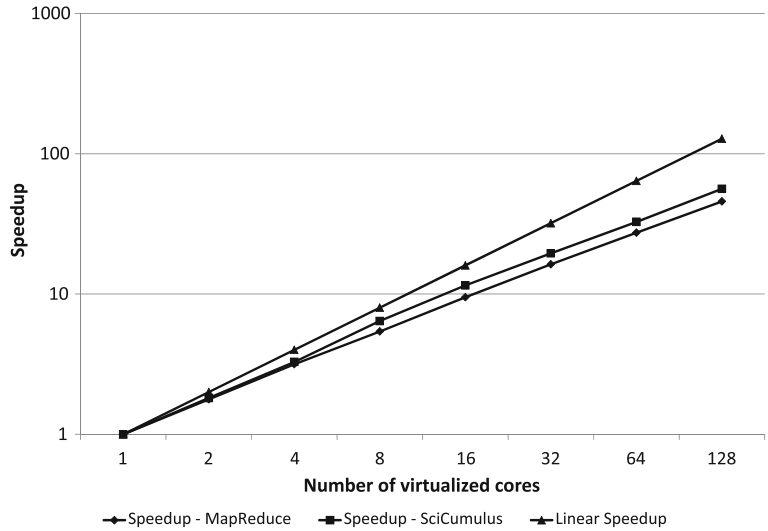

Fig. 11 Speedup for scenario S3

experiment consumed from 100 to 1,600 input files (multi-fasta files). The scalability results are presented in Fig. 14. As we can observe, as expected, that for each one of the scenarios (S1, S2, S3, S4 and Hadoop) as the volume of data to be consumed increases the execution time also increases. However, we can state that the overhead imposed by Hadoop in this case also scales up. Scenarios S1, S3 and S4 presented a good scalability by reaching a near linear scalability. However, scenario S2 could not scale in the same rate since its focus is on monetary cost instead of performance.

8.6 Financial Cost Analysis

In order to verify the financial cost involved in the execution of the five scenarios (Hadoop, S1, S2, S3 and S4), we analyzed generated provenance data and calculated the final cost using two payment forms. In the first form we set the quantum as 1 h (Amazon EC2 payment method) while in the second form we set the quantum as 1 min (GoGrid pre-paid payment form). We varied the type of used VM as explained in the last subsection. For each workflow execution, half of the instantiated

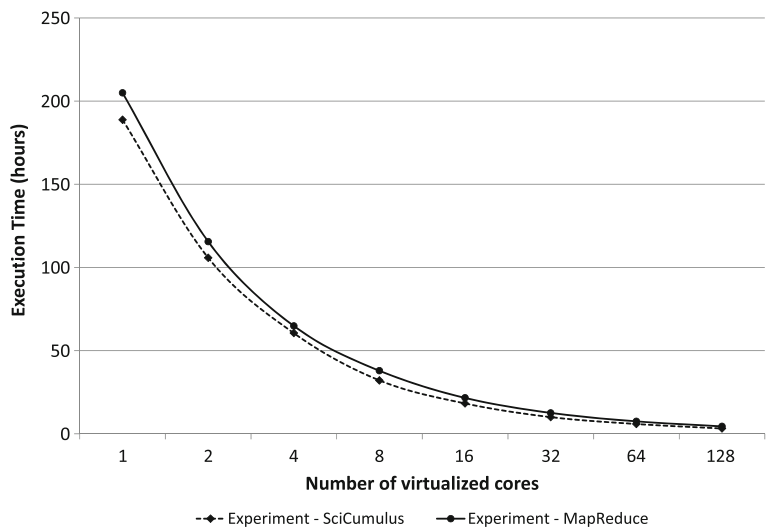

Fig. 12 Execution time for scenario S4

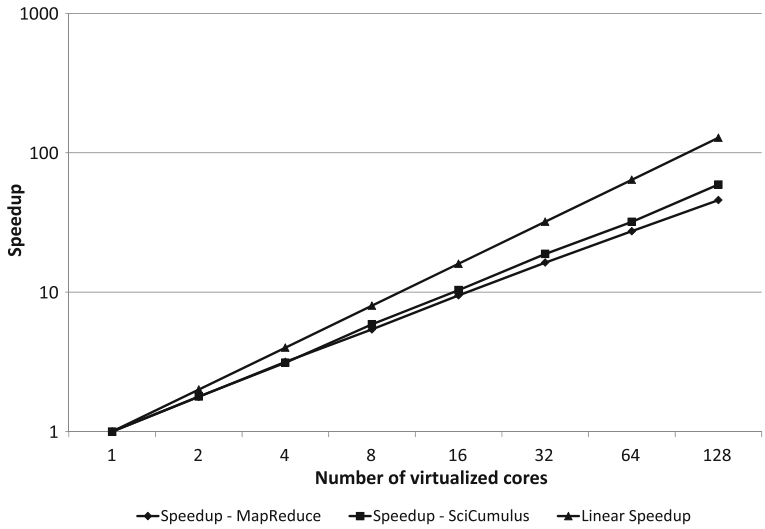

Fig. 13 Speedup for scenario S4

VMs is large ones and the other half is micro VMs. The overall financial cost did not vary much when executing four of five scenarios (Hadoop, S1, S3 and S4) because these executions are not focused on financial cost. However it is important to highlight that the achieved financial cost is not prohibitive, i.e. most of scientists are able to pay for those executions. We are more interested on analyzing the financial cost for the second scenario (S2) where we focused on reducing the financial cost. In this scenario the overall financial cost varies pretty much according with the chosen quantum as presented in Table 1 and Fig. 15.

This difference of financial cost is found because if we are executing the experiment in a provider in which the quantum is set to one minute we are going to schedule intensive cloud activities to VMs that are cheaper. This way, most of the execution time of the workflow is spent in micro VMs and just less intensive activities are executed in large VMs. Since less intensive activities execute in some minutes, the small quantum benefits because there is little wasted time. In addition, we pay only when we are executing a cloud activity. On the other hand, if we are using a one hour quantum, consider that a less

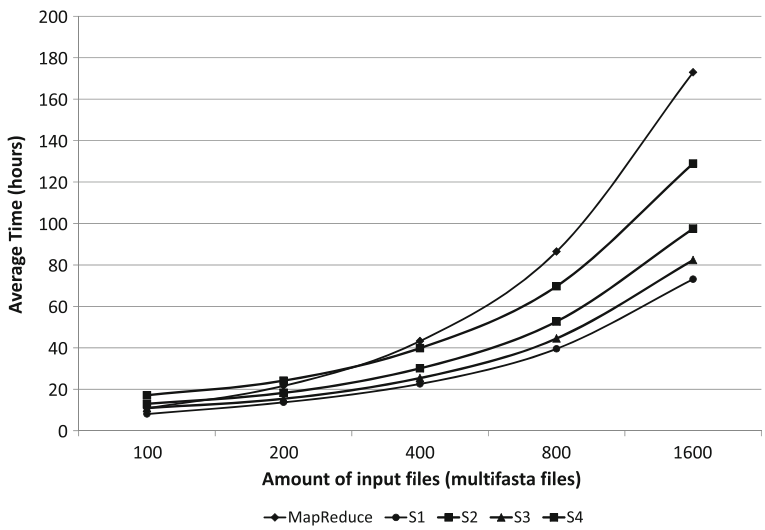

Fig. 14 Scalability according to the number of input files

Table 1 Total monetary cost for scenario S2

Number of VMs	Price (US$)—1 h quantum	Price (US$)—1 min quantum
1	66.15	65.15
2	82.79	37.61
4	107.23	48.72
8	125.65	57.09
16	135.18	61.44
32	185.80	84.66
64	201.47	91.58
128	256.07	116.35

intensive activity was scheduled to a large VM. This less intensive activity lasts for 16 min, but you are paying for one hour in this case, this way, wasting time and money. When the quantum is one minute if we spend some seconds of the quantum, the financial cost is not going to increase much. In addition, when the budget is about to be reached, the scheduler stops scheduling activities to expensive VMs to reduce the overall financial cost. This explains the difference in total cost when executing with 64 and 128 virtual cores.

8.7 Adaptive Performance Analysis

To analyze the performance of the adaptive approach in SciCumulus, we conducted a series of real experiments. We are specifically interested in comparing system performance using the adaptive approach and using scheduling algorithm without scaling resources, measuring and analyzing the overhead imposed by the adaptive approach so as to verify its benefits. The executed experiments fixed the number of consumed input multi-fasta files in to 100 and used just the large VM type (m1.large).

The experiments presented in this sub-section investigate the performance of the proposed adaptive algorithm with the cost model and their implementation while compared with the ideal result. Apart from current MapReduce solutions, which are all static, we could not analyze the advantages of the proposed adaptive algorithm compared with the existing scheduling algorithms because none of them focus on scheduling scientific workflow activities in a cloud.

As an experiment to analyze the adaptive approach where the load balancing algorithm scales resources up and down, we executed SciPhy in several scenarios. One of these executions is presented in Fig. 16. In this particular case we aim at analyzing the performance by setting up the deadline as 10 h (36,000 s) and the limit budget as US$ 150.00. The focus on performance is seen in the cost model ($\alpha_1 = 0.9, \alpha_2 = 0.05$ and $\alpha_3 = 0.05$). In this execution the number of VMs increase and decrease over the time to meet the deadline and the budget. In addition, VMs can be destroyed without interference of SciCumulus. This scenario occurs when we use Amazon EC2 spot instances

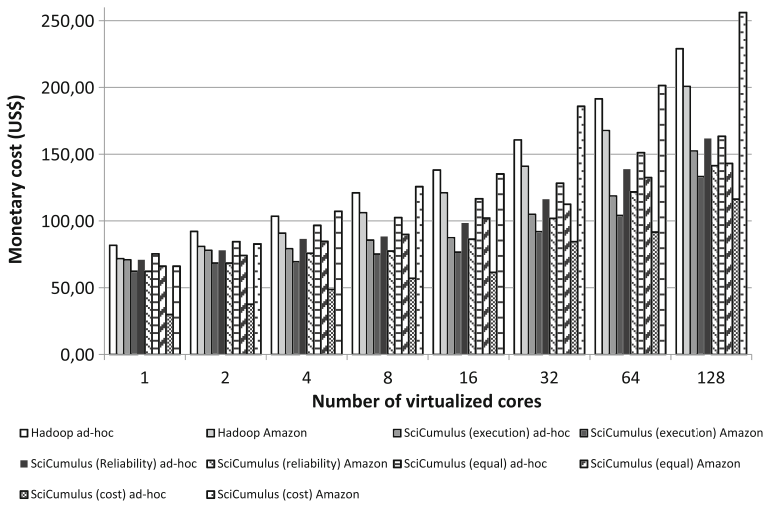

Fig. 15 Monetary cost analysis

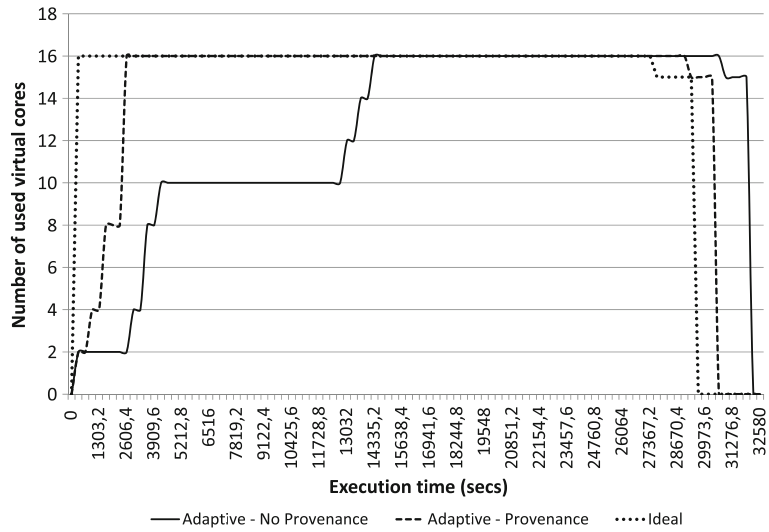

Fig. 16 Adaptive execution

(that can be destroyed by the cloud provider without previous warning).

We executed three different cases in this experiment. The first one we called "ideal scenario". We know a priori that if we executed SciPhy using 16 cores we could meet the deadline and the limit budget with the best possible performance. This way, the "ideal scenario" starts and ends using 16 cores. On the other hand, the other 2 cases, the amount of VMs increases and decreases gradually. In the first one we considered that there is no available provenance data. This way, the load balancing algorithm has to wait for performance provenance data to be generated to estimate the necessary amount of resources. In fact, the load balancing algorithm takes some time so start increasing the amount of VMs.

In the second one we assumed that there is provenance data available at the repository to perform a better estimation. This way, the load balancing algorithm starts to increase the number of VMs earlier than when there is no provenance available. The load balancing algorithm scales gradually because the first two activities of SciPhy are less intensive activities. When we reach the third activity (a compute intensive one) the

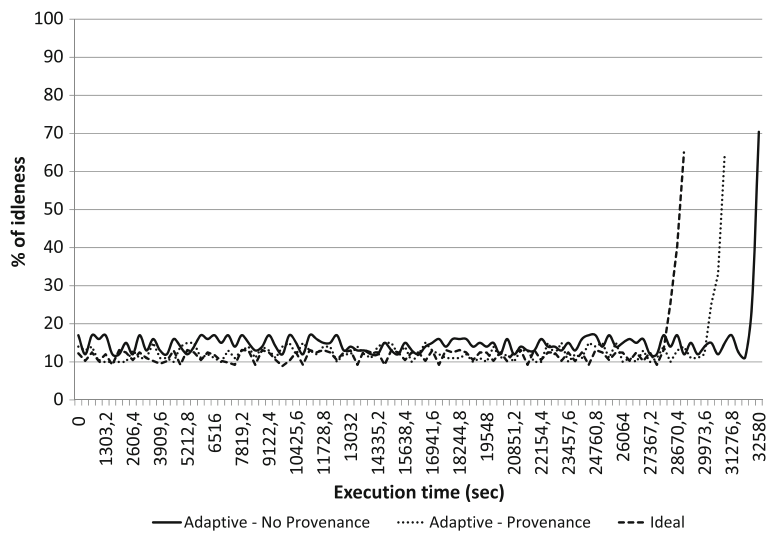

Fig. 17 Idleness analysis—adaptive execution

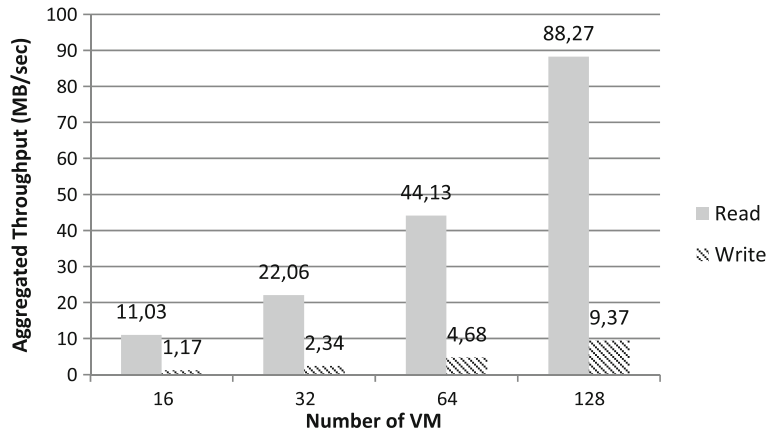

Fig. 18 Read and write throughput

algorithm scales the number of VMs faster. We can observe in Fig. 16 that the total amount of VMs increases and decreases gradually when using adaptive approach of SciCumulus. However, in this case the "static scenario" execution overcomes adaptive scheduling by 3.2 % and 8.1 % in the two scenarios where provenance is available and when it is not, respectively. These results are very near the ideal scenario, which is not known a priori by scientists. In addition, the adaptive approach avoids under and overestimations of the amount of resources to use.

We have also analyzed the level of idleness of the VMs during the workflow execution (Fig. 17). Since using the adaptive scheduling, several calculations have to be performed (cost model, estimation for scale number of VMs, etc.) SciCumulus takes more time to start a new cloud activity. This way, when using the adaptive approach, the level of idleness of the VMs tends to be greater than when using a "static scenario" approach. However, this difference is acceptable. The average idleness is 11.34 %, 12.60 % and 14.26 % when executing in a "static scenario", adaptive with provenance and adaptive without provenance, respectively.

8.8 Data Transferring Analysis

In order to understand the data transfer performance using SciCumulus in SciPhy workflow, we measured the throughput of SciCumulus components built on top of Amazon S3 framework. For this experiment we fixed the 200 input multi-fasta and large type of amazon VMs. Each VM reads and writes large volume of data from and to Amazon S3 environments (80 MB and 8.5 GB, respectively). The aggregated throughput is presented in Fig. 18. In this experiment, to transfer the 80 MB compressed files, it takes up to 7 min using a slow connection (256 mbps). This latency when compared to the overall execution time of SciPhy is acceptable. However, this input data can be already placed in S3, reducing the overall transferring cost. In addition, new mechanisms for data staging are planned to be developed in SciCumulus.

9 Conclusions

Large scale scientific experiments usually present a long duration where several executions, using different parameters and input data, are necessary to draw conclusions. Scientists frequently use parallel techniques to improve performance. However, it is far from trivial to manage parallel executions of large scientific workflows, particularly in cloud environments, which typically provide an elastic set of computing resources (VMs). To increase the uptake of the cloud model for executing scientific workflows that demand HPC capabilities, new solutions have to be developed, especially for scheduling parallel cloud activities in cloud resources.

Cloud activity scheduling is a well-known NP-complete problem even in its simplest form. Although there are several heuristic-based solutions to solve scheduling problems, most of them assume environments that do not change during the execution course of an experiment. These heuristics generate a priori scheduling plans (before starting executing the workflow). A priori scheduling is not an option when we execute scientific workflows in clouds. Since cloud resources are elastic, workflows should benefit from this elasticity. In addition, clouds are based on virtualization presenting fluctuations on the performance of VMs, thus impacting the overall workflow performance. This way, it is necessary to use an adaptive scheduling approach to consider those changes in the environment, including the number of VMs and the capacity of the VMs.

In previous work [16, 30, 61] we have addressed workflow execution in clouds using a static scheduling algorithm and simulated dynamic executions. One of the available infrastructures to execute scientific workflows in cloud environments is SciCumulus, a workflow execution engine to support parallel execution in clouds. To achieve high performance while keeping track of the workflow execution, SciCumulus has specific data provenance components to capture cloud execution information and drive the adaptive scheduling decisions at runtime.

This paper proposes a weighted cost function and an adaptive scheduling heuristic to explore the dynamicity of clouds while scheduling cloud activities to several VMs in a virtual cluster. Both the cost model and the scheduling algorithms were implemented in SciCumulus. Experimental results show that the proposed approach is able to find efficient solutions (depending on the chosen scenario—performance, financial cost, reliability and equal weight) when compared to an ideal solution and to other heuristics under evaluation (Hadoop approach).

Hadoop was the natural choice for a baseline, since we did not find scheduling algorithms that considered cloud elasticity issues and scientific workflow computations. This way, we have chosen to compare the static approach with Hadoop implementation, since many of the existing SWfMS parallel approaches are based on Hadoop. The adaptive approach was compared to the ideal solution, obtained artificially.

In the static analysis (where the number of VMs does not change during the course of the workflow) the overall performance results show that a near linear speedup was obtained when we executed SciPhy workflow processing 200 multifasta files and using from 2 up to 64 virtual cores. From 64 to 128 cores, the speedup presented some degradation. However, this degradation is acceptable. The static approach is useful when there is no (or a small) demand for HPC capabilities or when the environment is not susceptible to performance fluctuations, such as in a private (and controlled) cloud. The static approach is incompatible with environments that are elastic and whose processing capacity and the amount of resources vary over time.

The performance evaluation of the adaptive execution in SciCumulus showed results very close to the ideal solution. The evaluation also showed that the tradeoff between collecting provenance and performance prediction is very good. Only a small level of idleness was registered. Although the obtained results present a step forward, clouds are still beginning to be considered as a HPC scientific computation environment. There are only few approaches available and as clouds become more robust and more reliable, new techniques will be proposed. As future work, we plan to use the proposed approach in conjunction with a fault-tolerance mechanism named SciMultaneous that is focused on recovering failured cloud activities and execute them with more reliability on distributed cloud environments.

Acknowledgements We would like to thank CNPq, CAPES and FAPERJ for partially funding this work. We also thank Eduardo Ogasawara for his valuable comments with the experiments and Pedro Cruz together with Ricardo Busquet for their effort in developing SciCumulus components.

References

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

2. de Oliveira, D., Baião, F.A., Mattoso, M.: Towards a Taxonomy for Cloud Computing from an e-Science Perspective. In: Antonopoulos, N., Gillam, L. (eds.) Cloud Computing. Computer Communications and Networks, vol. 0, pp. 47–62. Springer, London (2010). doi:10.1007/978-1-84996-241-4_3
3. Foster, I., Kesselman, C.: The Grid: blueprint for a new computing infrastructure. Morgan Kaufmann, San Mateo, CA (2004)
4. El-Khamra, Y., Kim, H., Jha, S., Parashar, M.: Exploring the Performance Fluctuations of HPC Workloads on Clouds. In: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, 383–387 (2010)
5. Jackson, K.R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H.J., Wright, N.J.: Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, 159–168 (2010)
6. He, Q., Zhou, S., Kobler, B., Duffy, D., McGlynn, T.: Case study for running HPC applications in public clouds. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, 395–401 (2010)
7. Mattoso, M., Werner, C., Travassos, G.H., Braganholo, V., Murta, L., Ogasawara, E., Oliveira, D., da Cruz, S.M.S., Martinho, W.: Towards supporting the life cycle of large-scale scientific experiments. IJBPIM **5**(1), 79–92 (2010)
8. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: Workflows for e-Science: Scientific Workflows for Grids, 1 edn. Springer, Berlin Heidelberg New York (2007)
9. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-Science: an overview of workflow system features and capabilities. Future Gener. Comput. Syst. **25**(5), 528–540 (2009)
10. Davidson, S.B., Freire, J.: Provenance and scientific workflows: challenges and opportunities. In: ACM SIGMOD International Conference on Management of Data, pp. 1345–1350 (2008)
11. Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for computational tasks: a survey. Comput. Sci. Eng. **10**(3), 11–21 (2008)
12. Walker, E., Guiang, C.: Challenges in executing large parameter sweep studies across widely distributed computing environments. In: Workshop on Challenges of large applications in distributed environments, pp. 11–18 (2007)
13. Coutinho, F., Ogasawara, E., de Oliveira, D., Braganholo, V., Lima, A.A.B., Dávila, A.M.R., Mattoso, M.: Data parallelism in bioinformatics workflows using Hydra. In: 19th ACM International Symposium on High Performance Distributed Computing, pp. 507–515 (2010)
14. Jacob, J.C., Katz, D.S., Berriman, G.B., Good, J.C., Laity, A.C., Deelman, E., Kesselman, C., Singh, G., Su, M.-H., et al.: Montage: a Grid portal and software toolkit for science-grade astronomical image mosaicking. IJCSE **4**(2), 73–87 (2009)
15. Ogasawara, E., Oliveira, D., Chirigati, F., Barbosa, C.E., Elias, R., Braganholo, V., Coutinho, A., Mattoso, M.: Exploring many task computing in scientific workflows. In: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, pp. 1–10 (2009)
16. Oliveira, D., Ocana, K., Ogasawara, E., Dias, J., Baiao, F., Mattoso, M.: A performance evaluation of X-ray crystallography scientific workflow using SciCumulus. In: IEEE International Conference on Cloud Computing (CLOUD), pp. 708–715 (2011)
17. da Cruz, S.M.S., Batista, V., Dávila, A.M.R., Silva, E., Tosta, F., Vilela, C., Campos, M.L.M., Cuadrat, R., Tschoeke, D., et al.: OrthoSearch: a scientific workflow approach to detect distant homologies on protozoans. In: Proc. of the ACM SAC, pp. 1282–1286 (2008)
18. Oliveira, D., Ocaña, K.A.C.S., Ogasawara, E., Dias, J., Goncalves, J., Mattoso, M.: Cloud-based phylogenomic inference of evolutionary relationships: a performance study. In: Proceedings of the 2nd International Workshop on Cloud Computing and Scientific Applications (CCSA) (2012)
19. Ocaña, K.A.C.S., de Oliveira, D., Horta, F., Dias, J., Ogasawara, E., Mattoso, M.: Exploring molecular evolution reconstruction using a parallel cloud-based scientific workflow. In: Proceedings of the 2012 Brazilian Symposium on Bioinformatics (BSB 2012) (2012)
20. Ocaña, K.A.C.S., Oliveira, D., Ogasawara, E., Dávila, A.M.R., Lima, A.A.B., Mattoso, M.: SciPhy: a cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In: Norberto de Souza, O., Telles, G.P., Palakal, M. (orgs.) Advances in Bioinformatics and Computational Biology, pp. 66–70. Springer, Berlin (2011)
21. Al-Azzoni, I., Down, D.G.: Dynamic scheduling for heterogeneous Desktop Grids. In: 2008 9th IEEE/ACM International Conference on Grid Computing, pp. 136–143 (2008)
22. Smanchat, S., Indrawan, M., Ling, S., Enticott, C., Abramson, D.: Scheduling multiple parameter sweep workflow instances on the Grid. In: e-Science 2009—5th IEEE International Conference on e-Science, pp. 300–306 (2009)
23. Garg, S.K., Buyya, R., Siegel, H.J.: Scheduling parallel applications on utility Grids: time and cost trade-off management (2009)
24. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci. Program. **14**(3,4), 217–230 (2006)
25. Boeres, C., Sardiña, I., Drummond, L.: An efficient weighted bi-objective scheduling algorithm for heterogeneous systems. Parallel Comput. **37**(8), 349–364 (2011)
26. Qin, X., Hong, J.: A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. J. Parallel Distrib. Comput. **65**, 885–900 (2005)

27. Assayad, I., Girault, A., Kalla, H.: A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In: 2004 International Conference on Dependable Systems and Networks, pp. 347–356 (2004)
28. Amazon EC2. Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/ (2010)
29. Oliveira, D., Ogasawara, E., Baiao, F., Mattoso, M.: An adaptive approach for workflow activity execution in clouds. In: International Workshop on Challenges in e-Science—SBAC, pp. 9–16 (2010)
30. Oliveira, D., Ogasawara, E., Baião, F., Mattoso, M.: SciCumulus: a lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In: 3rd International Conference on Cloud Computing, pp. 378–385 (2010)
31. Lima, A., Mattoso, M., Valduriez, P.: Adaptive virtual partitioning for OLAP query processing in a database cluster. JIDM **1**(1), 75–88 (2010)
32. Kotowski, N., Lima, A.A.B., Pacitti, E., Valduriez, P., Mattoso, M.: Parallel query processing for OLAP in Grids. CCPE **20**(17), 2039–2048 (2008)
33. Paes, M., Lima, A.A.B., Valduriez, P., Mattoso, M.: high-performance query processing of a real-world OLAP Database with ParGRES. In: High Performance Computing for Computational Science (VECPAR), pp. 188–200 (2008)
34. Freedman, D., Pisani, R., Purves, R.: Statistics, 4th edn. W. W. Norton, New York (2007)
35. Dean, J., Ghemawat, S.: MapReduce: a flexible data processing tool. Commun. ACM **53**, 72–77 (2010)
36. Foster, I.: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison Wesley, Reading, MA (1995)
37. Wang, J., Crawl, D., Altintas, I.:. Kepler + Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In: 4th Workshop on Workflows in Support of Large-Scale Science, pp. 1–8 (2009)
38. Howe, B., Vo, H., Silva, C., Freire, J.: Query-driven visualization in the cloud with mapreduce. In: Proceedings of the Fourth Annual Workshop on Ultrascale Visualization (2009)
39. Lin, C., Lu, S.: Scheduling Scientific Workflows Elastically for Cloud Computing. In: 2011 IEEE International Conference on Cloud Computing (CLOUD), pp. 746–747 (2011)
40. Abramson, D., Enticott, C., Altinas, I.: Nimrod/K: towards massively parallel dynamic Grid workflows. In: Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–11 (2008)
41. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S.: Kepler: an extensible system for design and execution of scientific workflows. In: Scientific and Statistical Database Management, pp. 423–424 (2004)
42. Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J.: On the use of cloud computing for scientific workflows. In: IEEE Fourth International Conference on eScience (eScience 2008), Indianapolis, USA, pp. 7–12 (2008)
43. Deelman, E., Mehta, G., Singh, G., Su, M.-H., Vahi, K.: Pegasus: Mapping Large-Scale Workflows to Distributed Resources. In: Workflows for e-Science, pp. 376–394. Springer, Berlin Heidelberg New York (2007)
44. Warneke, D., Kao, O.: Nephele: efficient parallel data processing in the cloud. In: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, pp. 8:1–8:10 (2009)
45. Lee, C., Suzuki, J., Vasilakos, A., Yamamoto, Y., Oba, K.: An evolutionary game theoretic approach to adaptive and stable application deployment in clouds. In: Proceeding of the 2nd workshop on Bio-inspired algorithms for distributed systems, pp. 29–38 (2010)
46. Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., Mattoso, M.: An algebraic approach for data-centric scientific workflows. In: Proc. of VLDB Endowment, vol. 4, no. 12, pp. 1328–1339 (2011)
47. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 3rd edn. Springer, New York (2011)
48. Kllapi, H., Sitaridi, E., Tsangaris, M.M., Ioannidis, Y.: Schedule optimization for data processing flows on the cloud, 289 (2011)
49. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)
50. Meyer, L.A.V.C., Rössle, S.C., Bisch, P.M., Mattoso, M.: Parallelism in Bioinformatics Workflows. In: High Performance Computing for Computational Science—VECPAR 2004, pp. 583–597 (2005)
51. Callahan, S.P., Freire, J., Santos, E., Scheidegger, C.E., Silva, C.T., Vo, H.T.: VisTrails: visualization meets data management. In: SIGMOD International Conference on Management of Data, pp. 745–747 (2006)
52. Viana, V., de Oliveira, D., Mattoso, M.: Towards a cost model for scheduling scientific workflows activities in cloud environments. In: 2011 IEEE World Congress on Services (SERVICES), pp. 216–219 (2011)
53. Muniswamy-Reddy, K.-K., Macko, P., Seltzer, M.: Making a cloud provenance-aware. In: First workshop on on Theory and practice of provenance, pp. 1–10 (2009)
54. Simmhan, Y.L., Plale, B., Gannon, D.: A framework for collecting provenance in data-centric scientific workflows. ICWS, pp. 427–436 (2006)
55. Moreau, L., Freire, J., Futrelle, J., McGrath, R., Myers, J., Paulson, P.: The open provenance model: an overview. In: Provenance and Annotation of Data and Processes, pp. 323–326 (2008)
56. Greenwood, M., Goble, C., Stevens, R., Zhao, J. Addis, M., Marvin, D., Moreau, L., Oinn, T.: Provenance of e-Science Experiments—Experience from Bioinformatics. UK OST e-Science second All Hands Meeting **4**, 223–226 (2003)
57. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edn. Addison-Wesley Professional, Reading, MA (2003)

58. Shafi, A., Carpenter, B., Baker, M.: Nested parallelism for multi-core HPC systems using Java. J. Parallel Distrib. Comput. **69**(6), 532–545 (2009)
59. Gadelha, L.M.R., Mattoso, M.: Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems. In: International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES 2008), pp. 597–602 (2008)
60. SubCloud. Shared Enterprise File System for Amazon S3 Cloud Storage | SubCloud, http://www.subcloud.com/ (2011)
61. de Oliveira, D., Ogasawara, E., Ocaña, K., Baião, F., Mattoso, M.: An adaptive parallel execution strategy for cloud-based scientific workflows. Concurrency Computat.: Pract. Exper. **24**(13), 1531–1550 (2012). doi:10.1002/cpe.1880
62. Zvelebil, M., Baum, J.: Understanding Bioinformatics, 1 edn. Garland Science, New York (2007)
63. Miller, W., Makova, K.D., Nekrutenko, A., Hardison, R.C.: Comparative genomics. ARGHG **5**(1), 15–56 (2004)
64. Clark, A.G.: Genomics of the evolutionary process. Trends Ecol. Evol. **21**(6), 316–321 (2006)
65. Katoh, K., Toh, H.: Recent developments in the MAFFT multiple sequence alignment program. Brief. Bioinform. **9**(4), 286–298 (2008)
66. Katoh, K., Toh, H.: Parallelization of the MAFFT multiple sequence alignment program. Bioinformatics (Oxford, England) **26**(15), 1899–1900 (2010)
67. Lassmann, T., Sonnhammer, E.L.L.: Kalign–an accurate and fast multiple sequence alignment algorithm. BMC Bioinformatics **6**, 298 (2005)
68. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res. **22**(22), 4673–4680 (1994)
69. Do, C.B., Mahabhashyam, M.S.P., Brudno, M., Batzoglou, S.: ProbCons: probabilistic consistency-based multiple sequence alignment. Genome Res. **15**(2), 330–340 (2005)
70. Keane, T.M., Creevey, C.J., Pentony, M.M., Naughton, T.J., Mclnerney, J.O.: Assessment of methods for amino acid matrix selection and their use on empirical data shows that ad hoc assumptions for choice of matrix are not justified. BMC Evol. Biol. **6**, 29 (2006)
71. Stamatakis, A.: RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics (Oxford, England) **22**(21), 2688–2690 (2006)
72. Dutilh, B.E., van Noort, V., van der Heijden, R.T.J.M., Boekhout, T., Snel, B., Huynen, M.A.: Assessment of phylogenomic and orthology approaches for phylogenetic inference. Bioinformatics **23**(7), 815–824 (2007)
73. Apache Software Foundation. Hadoop. Internet Website, hadoop.apache.org/. Last accessed May 2009
74. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
75. Fei, X., Lu, S., Lin, C.: A MapReduce-Enabled Scientific Workflow Composition Framework. ICWS, pp. 663–670 (2009)
76. Hadoop. Apache Hadoop Web page, http://hadoop.apache.org/ (2012)
77. Carpenter, B., Getov, V., Judd, G., Skjellum, A., Fox, G.: MPJ: MPI-like message passing for Java. CCPE **12**(11), 1019–1038 (2000)
78. Pruitt, K.D., Tatusova, T., Klimke, W., Maglott, D.R.: NCBI Reference Sequences: current status, policy and new initiatives. Nucleic Acids Res. **37**(Database issue), D32–D36 (2009)