ExLibris RapidILL

Rapid #: -21269370

CROSS REF ID:	**6274823**
LENDER:	**AZN (Northern Arizona University) :: Ejournals**
BORROWER:	**HUL (Harvard University) :: Widener Library**
TYPE:	Article CC:CCL
JOURNAL TITLE:	Future generation computer systems
USER JOURNAL TITLE:	Future generations computer systems : FGCS.
ARTICLE TITLE:	Optimizing virtual machine allocation for parallel scientific workflows in federated clouds
ARTICLE AUTHOR:	Coutinho, Rafaelli de C.
VOLUME:	46
ISSUE:	
MONTH:	
YEAR:	2015
PAGES:	51-68
ISSN:	0167-739X
OCLC #:	
PATRON NOTES:	Thank you. 46

Processed by RapidX: 9/15/2023 1:10:15 PM

This material may be protected by copyright law (Title 17 U.S. Code)

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Optimizing virtual machine allocation for parallel scientific workflows in federated clouds

Rafaelli de C. Coutinho*, Lúcia M.A. Drummond, Yuri Frota, Daniel de Oliveira

Institute of Computing – Fluminense Federal University – Niterói, Brazil

HIGHLIGHTS

- We introduce an estimation of the amount of VMs to allocate in scientific workflows.
- The GraspCC-fed heuristic based on GRASP is proposed.
- GraspCC-fed considers scientific workflows in single-provider and federated clouds.
- An evaluation of GraspCC-fed is provided using SciPhylomics and adapted SciCumulus.
- GraspCC-fed is suitable to determine an optimal configuration for virtual clusters.

ARTICLE INFO

Article history:
Received 1 February 2014
Received in revised form
2 October 2014
Accepted 8 October 2014
Available online 22 October 2014

Keywords:
Cloud computing
Federated cloud
Parallel scientific workflow
Virtual machine allocation

ABSTRACT

Cloud computing has established itself as an interesting computational model that provides a wide range of resources such as storage, databases and computing power for several types of users. Recently, the concept of cloud computing was extended with the concept of federated clouds where several resources from different cloud providers are inter-connected to perform a common action (*e.g.* execute a scientific workflow). Users can benefit from both single-provider and federated cloud environment to execute their scientific workflows since they can get the necessary amount of resources on demand. In several of these workflows, there is a demand for high performance and parallelism techniques since many activities are data and computing intensive and can execute for hours, days or even weeks. There are some Scientific Workflow Management Systems (SWfMS) that already provide parallelism capabilities for scientific workflows in single-provider cloud. Most of them rely on creating a virtual cluster to execute the workflow in parallel. However, they also rely on the user to estimate the amount of virtual machines to be allocated to create this virtual cluster. Most SWfMS use this initial virtual cluster configuration made by the user for the entire workflow execution. Dimensioning the virtual cluster to execute the workflow in parallel is then a top priority task since if the virtual cluster is under or over dimensioned it can impact on the workflow performance or increase (unnecessarily) financial costs. This dimensioning is far from trivial in a single-provider cloud and specially in federated clouds due to the huge number of virtual machine types to choose in each location and provider. In this article, we propose an approach named GraspCC-fed to produce the optimal (or near-optimal) estimation of the amount of virtual machines to allocate for each workflow. GraspCC-fed extends a previously proposed heuristic based on GRASP for executing standalone applications to consider scientific workflows executed in both single-provider and federated clouds. For the experiments, GraspCC-fed was coupled to an adapted version of SciCumulus workflow engine for federated clouds. This way, we believe that GraspCC-fed can be an important decision support tool for users and it can help determining an optimal configuration for the virtual cluster for parallel cloud-based scientific workflows.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Scientists, in their daily duties, commonly deal with several flows of programs and data much larger than they can manually manage [1]. To assist scientists in managing all steps involved in large-scale experiments, scientific workflows have gained much

* Corresponding author.
E-mail addresses: rcoutinho@ic.uff.br (R.d.C. Coutinho), lucia@ic.uff.br (L.M.A. Drummond), yuri@ic.uff.br (Y. Frota), danielcmo@ic.uff.br (D. de Oliveira).

http://dx.doi.org/10.1016/j.future.2014.10.009
0167-739X/© 2014 Elsevier B.V. All rights reserved.

interest [2,3]. Many of these scientific workflows are large-scale and demand high performance computing (HPC) environments and parallel techniques to execute in a feasible time [2–4]. Thus, existing Scientific Workflow Management Systems (*i.e.* SWfMS) have to manage the execution of scientific workflows in a distributed and parallel forms in HPC environments. There are several examples of SWfMS available for executing workflows in HPC environments such as Swift/Turbine [5], Pegasus [6], Chiron [7], SciCumulus [8,9], Tavaxy [10] and Triana [11]. Most of these SWfMS are developed to run on traditional HPC environments such as computer clusters and grids [12]. However, in several universities and research centers many researchers do not have access to clusters or grids due to financial limitations or their experiments are seasonal (executions are performed during 3 weeks in the year, for example) which do not justify the massive investment in hardware.

To overcome this infrastructure issue, in the last few years, scientists start migrating their scientific experiments to clouds where computing resources (infrastructure, platform, and software) are provided (by a cloud provider) to users on demand via Internet [13,14]. Since resources are provided for a huge number of users, they only pay for their use. To access these on demand services for a scientific workflow execution, users might have to sign a service-level agreement (*i.e.* SLA) with a cloud service provider concerning how much bandwidth, CPU, and memory they can use at any given time.

Moreover, if scientific workflows need more computing power than one cloud provider is capable of offering they can benefit from a more elastic environment such as a federated cloud [15,16]. A federated cloud is a distributed and heterogeneous environment that aggregates infrastructures of several different cloud providers that may come from both the commercial and the academic area. In a federated cloud, the resources from multiple cloud providers (*e.g.* Amazon [17], Google [18], Microsoft [19]) are integrated to increase elasticity and reliability, thus being potentially beneficial for large-scale scientific workflows. For example, if a specific workflow execution demands 150 virtual machines to execute for 1 h, users cannot use only Amazon Elastic Compute Cloud (EC2) since Amazon EC2 limits the total number of allocated virtual machines to 20. (However, you can complete the Amazon EC2 instance form to request more than 20 On-Demand Instances[1].) Other reasons to use federated cloud are: (i) the size and the location of the data: if a workflow consumes extremely large data, it would be interesting not to transfer this data among the servers using a regular Internet connection. For example, consider a workflow for astronomy [20] where each input data has more than 5 GB of data. It is impractical to transfer several files without harming the workflow execution. So, it is fundamental that the computation is taken to where the data is, following the MapReduce strategy; (ii) the VMs characteristics: a cloud provider can offer VMs with characteristics that are more suitable for a specific problem. This way, several VMs in more than one provider have been allocated for the executions of these cases. For example, in these executions some activities of the workflow can execute in Amazon EC2 and others on Microsoft Azure.

Several SWfMS [5,6,8,9] already bridge the gap between a single-provider cloud and scientific workflow management, such as SciCumulus [8,9], Pegasus [6], and specifically for bioinformatics, the combination of Taverna and Galaxy named Tavaxy [10]. Just a few such as the engine proposed by Jrad et al. [21] and the adapted version of SciCumulus (introduced in this article) support the execution of scientific workflows in federated cloud environments. One common characteristic of all these approaches is that a virtual cluster is created in the cloud (single-provider or federated) to execute the workflow in parallel. Besides managing the parallel execution of the workflow, these SWfMS are also responsible for capturing information about the workflow execution, namely provenance [22] and can be used to reproduce a specific experiment.

This way, although single-provider clouds and federated clouds may represent an excellent opportunity for executing parallel scientific workflows that demand HPC capabilities, there are still several barriers for their use [23]. One of the most important barrier is how dimensioning a virtual cluster in a federated cloud before a parallel workflow execution. This virtual cluster may be composed by a huge set of VMs, each one of a different type in a different cloud provider. Even when we are executing the workflow in a single provider, it is far from trivial to estimate the correct amount of VMs of each type to allocate. Cloud providers offer a huge number of types of VMs to be allocated on demand. Each type has its own characteristics such as bandwidth, CPU, memory and financial cost. Under or over estimations can produce negative impacts in the workflow performance and high financial costs.

In all existing SWfMS, scientists have to decide by their own on how they want to use cloud services: the amount of VMs to allocate, whether or not to scale up the amount of VMs and how to measure the financial cost of the entire workflow execution. Due to the huge number of VM types to consider before VM allocation, the cloud dimensioning process commonly becomes tedious and error-prone if performed manually. This scenario is even more complex if we consider workflows executed in federated clouds. In a federated cloud scenario, the number of VMs types to be considered for dimensioning the virtual cluster increases exponentially thus making the cloud dimensioning process impractical to be manually performed.

To overcome this barrier of dimensioning the virtual cluster for a workflow execution, some solutions were already introduced to be coupled to existing SWfMS such as SciDim [24]. SciDim is a service that aims at dimensioning the virtual cluster for a specific workflow execution. It does not require human intervention but can produce non-optimal estimations, since it is based on genetic algorithms [25–27]. This is not a problem for SciDim because it was designed to be coupled to adaptive engines such as SciCumulus that can add or remove virtual machines during workflow executions. However, most of cloud-based SWfMS are non-adaptive and this way non-optimal estimations may produce over and under dimensioning. In addition, SciDim is focused on dimensioning the virtual cluster in a single-provider cloud, thus not concerning about the execution in federated clouds, which is considerably more complex. Therefore, the estimation of the amount of virtual machines to be allocated to create a virtual cluster for parallel workflow in executions in both single-provider and federated clouds is an open, yet important, problem.

In this article, we claim that one way to manage (and to reduce) the total execution time of the workflow while controlling financial costs is to find the optimal amount of VMs to allocate for a specific workflow execution before its execution, independently if workflows are executed in a single-provider cloud or in a federated cloud. Of course, this initial allocation can be further refined by adaptive SWfMS (when they are available), but it already reduces the unnecessary allocation of virtual machines right from the start of the workflow execution. We also believe that this estimation must be performed by a specific tool that should be coupled to cloud-based SWfMS.

This way, we propose an approach named GraspCC-fed to be coupled to a prototype of SciCumulus workflow engine for federated clouds, named SciCumulus-fed. GraspCC-fed extends a previously proposed heuristic based on Greedy Randomized Adaptive Search Procedure (GRASP) [28] to consider scientific workflows executed in both single-provider and federated clouds. This

1 https://aws.amazon.com/ec2/faqs/.

extension considers that scientific workflows are executed in parallel using the First-Activity–First dataflow strategy as proposed by Ogasawara et al. [29]. From the scheduling perspective, FAF is not optimal, as it introduces additional constraints on workflow execution. The proposed approach is based on the actual way that popular cloud providers, such as Amazon EC2 and Microsoft Azure offer resources and services for users that need HPC capabilities. In order to evaluate it, the heuristic is compared with other similar approaches found in the literature such as the use of genetic algorithms for dimensioning the virtual cluster.

All performed experiments are based on a fragment of SciPhylomics [30], a large-scale scientific workflow for phylogenomic analysis and were executed in real commercial clouds presenting federated cloud scenarios. The obtained results indicate that SciPhylomics executing in virtual cluster dimensioned by GraspCC-fed lasted 59% less than the same workflow executed in a virtual cluster dimensioned by genetic algorithms in a single-provider cloud. In a federated cloud environment, despite some performance problems due to limitations on the prototype of the virtual cluster dimensioned by GraspCC-fed also executed SciPhylomics in less than 4.5 h. This way, we believe that GraspCC-fed can be an important decision support tool for users and it can help determining an optimal configuration for the virtual cluster in parallel cloud-based scientific workflows thus reducing the overall workflow execution time and financial cost.

The remaining of the article is organized as follows. Section 2 describes works that are related in some way to the cloud virtual machine allocation problem that we are addressing here. Section 3 presents a brief explanation of SciCumulus that is used to execute the workflow in single and federated cloud scenarios using the configurations provided by GraspCC-fed. In Section 5, we compare a GRASP-based approach with a genetic algorithm approach for single cloud dimensioning in order to choose the best approach to use for federated cloud dimensioning. Section 7 extends the GRASP-based approach for virtual machine allocation problem for federated clouds. Experimental results are reported in Section 8. Finally, Section 9 concludes the paper and points future work.

2. Related work

This section aims at presenting a review of the literature in the topic of virtual machine allocation for parallel scientific workflows in federated clouds. To the best of authors' knowledge, few papers in literature treat the virtual machine allocation problem specifically for parallel workflows and none of them is based on federated clouds. Due to this reason, we have chosen to present a broader view of this topic, thus first covering more general papers of virtual machine allocation in clouds and then presenting specific papers in the context of scientific workflows.

Endo et al. [31] categorize the main challenges inherent to the virtual machine allocation problem in compute clouds, offering a stepwise view of this problem that covers from the initial modeling phase to the optimization phase. Most of existing methods focus on optimizing communication and application execution time while others focus on energy-efficient virtual machine allocation.

For example, Meng et al. [32] propose the use of traffic-aware virtual machine placement (TVMPP) to improve the network scalability and formulate the virtual machine placement as an optimization problem and prove its hardness. The proposed algorithm has two design principles. The first one is to find a mapping of virtual machines to slots such that virtual machine pairs with heavy mutual traffic are assigned to slot pairs with low-cost connections. The second design principle is the application of the divide-and-conquer strategy, where virtual machines are partitioned into VM-clusters and slots into slot clusters.

Pandey et al. [33] formulate a model and design a scheduling heuristic that uses a particle swarm optimization (PSO) for task–resource mapping aiming to minimize the overall cost of execution, taking into account both computation and data transmission costs. A method to solve the virtual machine scheduling problem based on SLA in clouds is also proposed by Li and Guo [34]. This method applies stochastic integer programming for optimal virtual machine allocation in cloud computing and uses Gröbner bases theory [35] for solving the stochastic integer programming problem.

Zhao et al. [36] present a multi-objective optimization model for virtual machine allocation problem and solves it by an advanced Non-dominated Sorting Genetic Algorithm II (NSGA II). The proposed algorithm is composed of an objective function for each attribute of virtual resources, CPU, memory and bandwidth, and uses a classic algorithm for multi-objective optimization as the search method. Alicherry et al. [37] developed an algorithm for the optimal selection of data centers in clouds and use a similar algorithm to select, within each data center, the racks and servers where the requested VMs will be allocated. They propose a heuristic for partitioning the requested virtual machines among the chosen data centers and racks. Chaisiri et al. [38] propose an optimal cloud resource provisioning (OCRP) algorithm by formulating a stochastic programming model. The OCRP algorithm can provision virtual machines to be used in multiple provisioning and considers the demand and price uncertainty. Different approaches such as deterministic equivalent formulation, sample-average approximation and Benders decomposition are considered. Coutinho et al. [28] modeled a virtual machine allocation problem that selects cloud resources aiming to reduce the financial cost and the execution time of consumer applications as an integer programming problem. They propose a heuristic based on Greedy Randomized Adaptive Search Procedure (GRASP) to solve it.

There are also several papers about energy consumption of network and computing resources that underpin the cloud. An analysis of energy consumption in cloud computing is performed by Baliga et al. [39]. This analysis considers both public and private clouds, and includes energy consumption in switching and transmission as well as data processing and data storage. Many proposals can reduce the energy consumption of applications and, by transitivity, of cloud infrastructures, but they do not consider the internal characteristics of applications to define a trade-off between applications QoS and energy consumption. Oliveira et al. [40] propose a self-adaptation approach that considers both application and system characteristics in virtual machine allocation to reduce the energy consumption in cloud infrastructure. Beloglazov et al. [41] propose architectural principles for energy-efficient virtual machine allocation in clouds and scheduling algorithms considering QoS expectations and power usage characteristics of the devices. It also presents open research challenges, addressing which can bring substantial benefits to both resource providers and consumers. An efficient heuristic algorithm based on convex optimization and dynamic programming is presented by Goudarzi et al. [42] to solve the virtual machine allocation problem in clouds by minimizing the total energy cost of cloud computing system while meeting the specified SLA in a probabilistic sense. Goudarzi et al. [43] also propose a different approach for allocating virtual machines where the proposed approach consists of exploiting virtual machine copies to increase the service reliability and also to serve the requests. Generally, only the original copy of the VM handles the requests while the other copies are idle. The proposed algorithm creates at first multiple copies of VMs and then uses dynamic programming and local search to place these copies on the physical servers. Xu et al. [44] formulate an energy efficiency virtual machine allocation for clouds as a multi-objective optimization problem. The simulation results showed that the proposed approach is able to generate schedules with different numbers of servers and virtual machines with diverse characteristics in a reasonable time thus decreasing the total operating energy of data center.

Although the aforementioned approaches demonstrated suitability for different kinds of applications and purposes they cannot be directly applied to allocate virtual machines for a parallel scientific workflow execution. These approaches proved to be scalable and consider several important issues such as execution time, data placement, communication and financial issues. However, they do not consider specific workflow issues such as data dependency between activities and the behavior of the workflow execution neither federated cloud environments. On the other hand, there are some papers that focus on optimization of virtual machine allocation specifically for scientific workflows (executed in parallel or not). One example is proposed by Malawski et al. [45]. In this paper the authors propose a series of adaptive scheduling algorithms for cloud-based scientific workflows that provides dimensioning and vertical scaling during workflow execution to comply with users' constraints. Performance results presented by Malawski et al. are based on simulations of several workflow executions. Although this approach represents a step forward, it does not take into account the optimization of the initial virtual machine allocation, *i.e.* it brings opportunities for adjusting the amount of virtual machines before the execution of scientific workflow, according to the constraints defined by the users. We believe that the performance of these adaptive virtual machine configurations can be improved if the virtual machine allocation is optimized before the workflow execution. Zhao et al. [46] propose an approach to manage workflow execution in the cloud using Swift [47] SWfMS. This approach has a specific component for on-demand virtual cluster provisioning (*i.e.* virtual machine allocation). Actually Zhao et al. approach is based on allocation algorithms of OpenNebula [48] that is a toolkit to build Infrastructure-as-a-Service (IaaS) clouds. OpenNebula provides automatic initial virtual machine allocation for the definition of workload and resource-aware allocation policies such as packing, striping, load-aware, and affinity-aware. However, Zhao et al. do not present how these algorithms should be adapted to be applied in the context of the parallel workflow execution. Zhu et al. [49] propose a framework named pSciMapper that aims at allocating virtual machines in the cloud for scientific workflow execution. They consider the virtual machine allocation as a hierarchical clustering problem, and they introduce a distance metric that is based on relationship between virtual machine requirements. Actually, their goal is to group workflow activity executions that are similar in their resource requirements and they allocate virtual machines for each group. They used the Nelder–Mead [50] optimization algorithm for that. However, pSciMapper does not take into account data transfer issues neither executing the workflow using federated clouds. Deng et al. [51] propose a model to explore the virtual machine allocation problem for a sequential scientific workflow. Based on the proposed model, they developed a scheduler that includes multiple allocation policies. They also propose an algorithm that enlarges the chance of selecting the best policy in limited time, *e.g.* runtime. However, they use a greedy algorithm in this paper that can lead to near-optimal solutions instead of optimal solutions. In addition, they do not consider the execution of the workflow in a federated cloud scenario. Lama et al. [52] propose AROMA, an approach that automatically allocates virtual machines for MapReduce jobs in the cloud. Although MapReduce-like approaches are not similar to SWfMS, the approach can be extrapolated to the workflow domain since MapReduce jobs can be also used to model workflows as presented by Nguyen and Halem [53], Wang et al. [54] and Crawl et al. [55]. AROMA uses vector machines and genetic algorithms to estimate the necessary amount of resources and then allocate the virtual machines. Although AROMA proved its effectiveness in providing performance guarantee of diverse Hadoop jobs, it does not consider financial issues neither federated cloud environments. Mandal et al. [56] propose a heuristic strategy using GRASP for optimal load-balancing on grid resources using the GrADS [57] infrastructure. This approach could be also extrapolated for cloud environments. However, since the original approach is focused on grids, they do not consider financial costs neither federated cloud environments. Oliveira et al. [24] modeled a multi-objective cost function allied to genetic algorithms in a service called SciDim to optimize virtual machine allocations in clouds. The SciDim uses provenance data to help determining an ideal initial configuration for the virtual cluster, under budget and deadline constraints set by the users. Although SciDim considers both performance and financial issues in scientific workflows, it does not take into account the federated cloud scenario.

To the best of our knowledge, none of the aforementioned approaches have investigated the virtual machine allocation problem for workflows that are executed in federated cloud environments. To propose a solution that can be applied to the federated cloud scenario we have chosen to investigate two of the most prominent metaheuristics to verify which one is more suitable for estimating the amount of virtual machines to allocate in federated clouds. This way, GRASP and Genetic Algorithm (implemented in SciDim) were elected and both approaches are described and compared in detail in Section 5.

3. SciCumulus engine

This section presents a brief explanation of SciCumulus cloud workflow engine architecture. Since the proposed approach for virtual machine allocation is going to be coupled to SciCumulus, it is important to provide a brief explanation about the engine in this article. SciCumulus is an essential component of this article since it provides an adaptive scheduler for parallel scientific workflows to be combined with the proposed virtual machine allocation approach. It is necessary to highlight that other cloud workflow execution engines, besides SciCumulus, can also benefit from the proposed approach, particularly non-adaptive approaches such as Swift/Turbine and Pegasus.

3.1. Cloud activity concept

Before starting explaining SciCumulus architecture we have to explain a fundamental concept that is *cloud activity* (CA). In SciCumulus, activities within a workflow are ruled by algebraic operators that consume and produce sets of tuples (relations) as proposed by Ogasawara et al. [29]. Since in a parallel workflow execution the same activity can execute several times varying parameters and input data, SciCumulus has to manage these several executions. To support such executions, SciCumulus is based on the concept of CA, inspired by database tuple activations [58]. The concept of CA is critical to make a workflow representation able to execute in large-scale cloud environments. A CA is a self-contained object that stores all information needed for a specific execution (*i.e.* the program to invoke and the portion of data to consume) to execute an activity at any virtual machine. CAs contain the finest unit of data needed by an activity to execute [58]. CAs are processed in SciCumulus following a 3-step procedure: (i) instrumentation where the parameter values are gathered to prepare the program invocation, (ii) program invocation where the executable code is effectively executed, and (iii) output extraction where result data is extracted and stored in the provenance database [59].

Fig. 1 presents a theoretical example of the CA generation. In this example we start the workflow with a split activity and

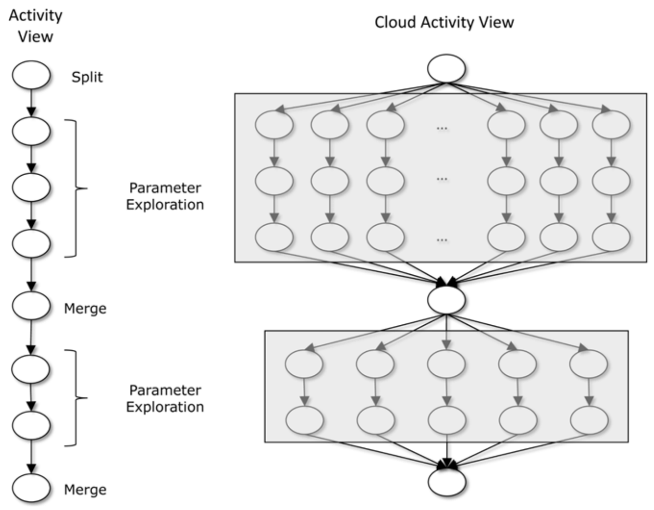


Fig. 1. Example of CA generation [59].

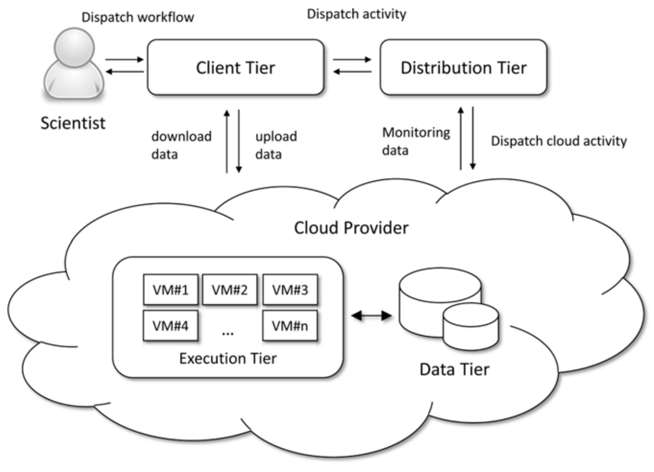


Fig. 2. SciCumulus conceptual architecture [59].

we have also two merge activities (Activity View). These three activities cannot be executed in parallel. However, the other activities can execute in parallel, thus creating a different CA for each different parameter value to consume (Cloud Activity View). Each generated CA is then scheduled to execute in a specific VM of the several allocated virtual machines.

3.2. Architecture

SciCumulus is an engine that manages the parallel execution of scientific workflows in clouds, such as Amazon EC2 [17] or Microsoft Azure [19]. SciCumulus architecture is composed by four tiers: Client tier, Distribution tier, Execution tier and Data tier. A high-level conceptual architecture is summarized in Fig. 2. For more information about SciCumulus please refer to Oliveira et al. [8,59,60].

SciCumulus client tier is responsible for initiating the execution of workflow activities in the cloud. The components of this tier are deployed in the user desktop or within a non-parallel SWfMS, such as VisTrails [61] or Kepler [62]. The distribution tier manages the execution of the several CAs. This tier is responsible for creating the scheduling plan for the CAs to be executed in the different virtual machines that are part of the execution tier. The execution tier is responsible for invoking executable codes (*i.e.* programs that are part of the workflow) in the several allocated virtual machines. Finally, the data tier has the provenance repository and the file system that stores all files produced and consumed during workflow execution.

Differently from other SWfMS, SciCumulus is able to execute either in static or adaptive [59] execution modes. In the static mode, SciCumulus schedules the several CAs based on the allocated virtual machines in the beginning of the workflow execution, *i.e.* the amount of virtual machines does not vary on the workflow execution course. In the adaptive mode, SciCumulus performs a horizontal scaling according to the cloud environment changes and availability of resources in order to maintain the overall performance of the workflow. This way, SciCumulus is able to add (or remove) virtual machines into (or from) the existing pool of resources. Although the adaptive mode is a step forward it introduces overhead in the execution [59]. This way, if the number of allocated virtual machine is optimal or near optimal (the amount of virtual machines that provides the best performance or the minimum financial cost following users' criteria) in the beginning of the workflow execution, it could spare significant amount of time.

3.3. Dataflow strategies

In this subsection we briefly present the two possible dataflow strategies used in SciCumulus. These strategies are based on the strategies originally proposed by Ogasawara et al. for Chiron workflow engine [7]. The presented dataflow strategies define how SciCumulus distributes CAs. A *First Tuple First* (or simply FTF) strategy partitions a set of CAs of a specific activity into a set of strict schedules for each portion of data. For example, in Fig. 3(a) we have a workflow composed by activities A, B and C and each one has three associated CAs. In the FTF strategy as soon as CA a_1 finishes we can start CA b_1 (which has a data dependency) without having to wait for the other CAs of A finish. On the other hand, in the *First Activity First* strategy we can only start the CAs of B when all CAs of A are finished, *i.e.* in Fig. 3(b) if a_1 finishes, we can only execute b_1 when a_2 and a_3 are also finished. The FAF strategy is fundamental for this article since it is the basis of the proposed approach.

From the scheduling perspective, FAF is not optimal, as it introduces additional constraints on workflow execution. Even if each activity is scheduled optimally, the schedule might not be globally optimal. For instance, consider the example from Fig. 3, with the following CA durations: $a1 = b2 = c3 = 3$; remaining CAs $= 1$. A FTF schedule on 3 machines has an optimal makespan of 5; a FAF schedule has a makespan of 9.

4. SciPhylomics: a scientific workflow for phylogenomic analysis

This section presents the SciPhylomics workflow that was used as a case study to compare GraspCC and SciDim. In the last few years, phylogenomic analysis experiments are evolving in a fast pace due to new technologies such as new DNA-sequencing methods and scientific apparatus [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 phylogenomic experiments is not a trivial task to be accomplished since they are compute intensive and process large amounts of data. As they are based on a pipeline of scientific programs, phylogenomic experiments can be modeled as scientific workflows. In this workflow, biologists perform a parameter sweep in a set of activities to produce a (commonly large) set of phylogenetic trees, which are used to infer evolutionary relationships between genome of divergent species.

In order to model a phylogenomic analysis experiment as a scientific workflow it was proposed SciPhylomics [59]. SciPhylomics

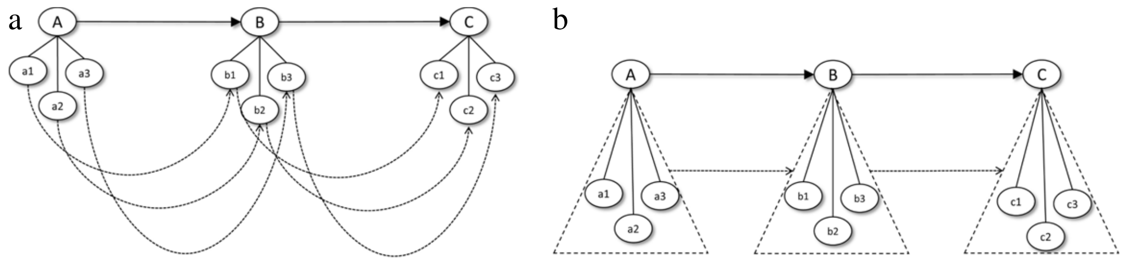


Fig. 3. (a) FTF dataflow strategy; (b) FAF dataflow strategy.

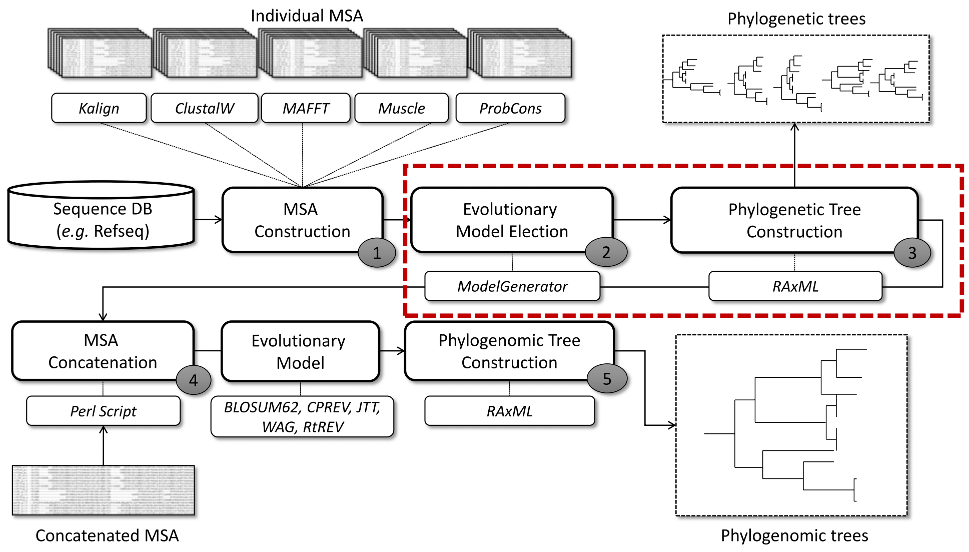


Fig. 4. Conceptual view of Sciphylomics. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
Source: Adapted from Oliveira et al. [59].

is a parameter sweep workflow where the same set of activities is executed for each file in a given large input dataset. It is composed by five main activities as presented in Fig. 4. The first three activities are: (i) multiple sequence alignment (MSA), this activity can be executed by a set of programs such as MAFFT [65,66], Kalign [67], ClustalW [68], Muscle [69] and ProbCons [70] and each MSA program receives a multifasta file as input, then producing a MSA as output; (ii) search for the best evolutionary model—each generated MSA is tested to find the best evolutionary model using ModelGenerator [71] program; and (iii) construction of phylogenetic trees uses RAxML [72] to generate phylogenetic trees using RAxML with 100 bootstrap replicates [63]. Consequently, we obtain several trees for each one of the MSA programs. In phylogenomic analysis, there is the option to concatenate multiple gene sequences to construct phylogenetic trees on the genomic level, also known as “genome trees”, “superalignment 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. This way, in the fourth activity, all individual MSA are used as input for a Perl script, in which they are concatenated thus obtaining a superalignment [73] 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 using RAxML as well.

Since biologists aim at executing a parameter sweep in SciPhylomics, 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 CA (also called blocking activity) and just one instance of it is allowed per MSA program. The evolutionary 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 SciPhylomics can be viewed in Fig. 5. Each one of the circles represents a different CA that is scheduled and executed in parallel in the several allocated virtual machines. Since an execution of SciPhylomics may last for days and weeks, in this article we have chosen to use a compute intensive fragment of the workflow as presented in Fig. 4 in the dashed red line. ModelGenerator and RAxML are compute intensive programs and their performance directly impacts the overall workflow performance.

5. Choosing the approach

In the papers published from 2007 until nowadays we can find many different goals of Cloud Computing technology as stressed by Vaquero et al. [14]. Although most of the goals are indeed important, in this article, we consider only the Infrastructure-as-a-Service (Iaas) model where users, that need to execute large-scale scientific workflows, request computing resources (*i.e.* virtual machines) for a period of time and pay only for what they use. However, allocating the necessary amount of virtual machines for a specific workflow execution is far from trivial because it is hard for users to estimate the execution time of activities, the amount of data files produced and associated transfer times. Actually, if the amount of virtual machines to be allocated is under or over dimensioned it surely produces negative impacts on the performance of the workflow or the financial costs of the execution. This allocation task is very complex because cloud providers commonly have a wide variety of virtual machine

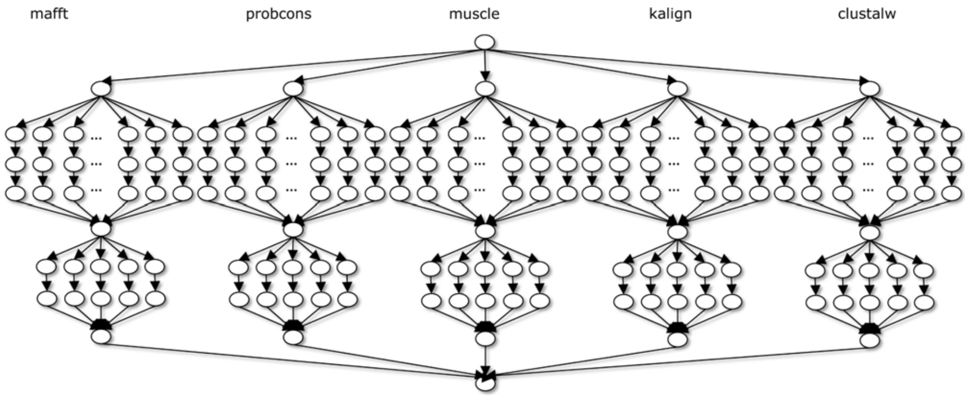


Fig. 5. Parameter sweep for SciPhylomics [59].

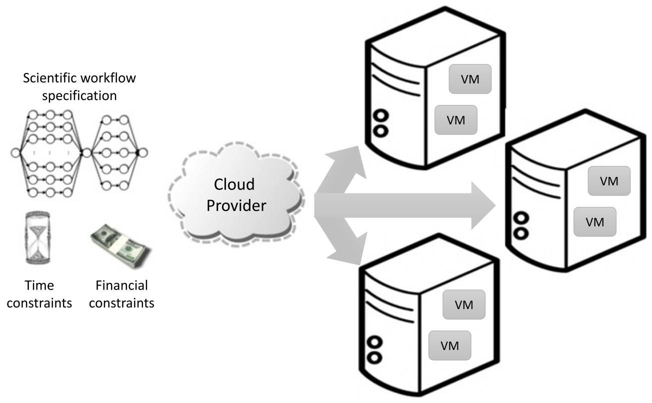


Fig. 6. Virtual machine allocation scenario [77].

types (*e.g.* micro, high performance CPU, computer clusters, GPU clusters, and high operations I/O), each one associated with a financial cost and performance characteristics, designed to meet different requirements of users or a common customer. In this complex scenario, users have to decide which types of virtual machine (and the correct amount) they should allocate in order to minimize a specific objective as execution time or financial cost. Fig. 6 illustrates this scenario, where users request virtual machines to a cloud provider to execute their workflows and the provider in turn offers a set of virtual machine types (CPU, memory and storage disk) to users. In this scenario, users can select more than one virtual machine type and these virtual machines can share resources of physical machines. To avoid under and over dimensioning of the cloud environment, we need to use a virtual machine allocation strategy that provides an optimal estimation. However there are many possible approaches for cloud dimensioning. In this way we investigated two approaches in order to select the best one for modeling federated clouds (which are more complex). In the following, we describe and compare two different approaches that are used to estimate the necessary amount of virtual machines to allocate for parallel workflow executions: a GRASP-based and a Genetic Algorithm-based approach. These approaches were chosen since they are highly used for solving allocation problems in scientific workflows and in other domains [74–76]. The approach that best performed for a single-provider cloud execution is then used for modeling the virtual machine allocation problem for the complex scenario of scientific workflows executed in federated clouds.

5.1. Problem definition

The aforementioned virtual machine allocation scenario can be described as a mathematical formulation as previously presented by Coutinho et al. [28] where the grounds for the entire notation used from here on are set. Let P be the set of virtual machine types offered by a cloud provider during a set of time periods. A set of users' requirements such as the maximum financial cost C_M, maximum execution time T_M, disk storage D_S, memory capacity M_C and a processing demand of G_f Gflops are defined. Similarly, each virtual machine type $p \in P$ has an associated financial cost c_p (*i.e.* the cost of purchasing the virtual machine for one period of time, the periods of time vary according to cloud provider, it can be 1 h or 1 min) and computing resources as disk storage d_p, memory capacity m_p and a processing power of g_p Gflop per period of time (Gflopt). Moreover, it is common that cloud providers have a maximum limit of N_M allocated virtual machines for each user in each period of time (see Table 1 in the description of used notation). Note that, the availability of virtual machines is not considered here.

A binary variable x_{pit} for each $p \in P, i \in \{1, \ldots, N_M\}$ and $t \in T = \{1, \ldots, T_M\}$, such that $x_{pit} = 1$ if and only if virtual machine i of type p is allocated at time t, otherwise $x_{pit} = 0$ is also defined. Also, consider variable t_m as the last time that a virtual machine was allocated by the user. In scheduling, this variable is also called the makespan. The described scenario can be formulated as follows:

$$\text{(CC-IP)} \quad \min \left(\alpha_1 \sum_{p \in P} \sum_{i=1}^{N_M} \sum_{t \in T} c_p x_{pit} + \alpha_2 t_m \right) \tag{1}$$

$$\text{subject to} \quad \sum_{p \in P} \sum_{i=1}^{N_M} \sum_{t \in T} c_p x_{pit} \le C_M \tag{2}$$

$$\sum_{p \in P} \sum_{i=1}^{N_M} d_p\, x_{pit} \ge D_S\, x_{p'i't}, \qquad \forall t \in T, \forall p' \in P, \forall i' \in \{1, \ldots, N_M\} \tag{3}$$

$$\sum_{p \in P} \sum_{i=1}^{N_M} m_p\, x_{pit} \ge M_C\, x_{p'i't}, \qquad \forall t \in T, \forall p' \in P, \forall i' \in \{1, \ldots, N_M\} \tag{4}$$

$$\sum_{p \in P} \sum_{i=1}^{N_M} \sum_{t \in T} g_p x_{pit} \ge G_f \tag{5}$$

$$\sum_{p \in P} \sum_{i=1}^{N_M} x_{pit} \le N_M, \quad \forall t \in T \tag{6}$$

$$t_m \ge t\, x_{pit}, \qquad \forall t \in T, \forall p \in P, \ \forall i \in \{1, \ldots, N_M\} \tag{7}$$

Table 1
Notations for clouds environment.

Notation	Description
P	The set of virtual machine types
C_M	The maximum financial cost requirement
T_M	The maximum execution time requirement
D_S	The disk storage requirement
M_C	The memory capacity requirement
G_f	The processing demand requirement
c_p	The cost of purchasing the virtual machine p for one period of time
d_p	The disk storage of virtual machine p
m_p	The memory capacity of virtual machine p
g_p	The processing power of virtual machine p
N_M	The maximum limit of allocated virtual machine for each user in each period of time
x_{pit}	$x_{pit} = 1$, if and only if virtual machine i of type p is allocated at time t
t_m	The last time period that a virtual machine was allocated by the user

$$x_{pit+1} \leq x_{pit}, \qquad \forall t \in T, \forall p \in P, \forall i \in \{1, \ldots, N_M\} \tag{8}$$

$$x_{pi+1t} \leq x_{pit}, \qquad \forall t \in T, \ \forall p \in P, \forall i \in \{1, \ldots, N_M - 1\} \tag{9}$$

$$x_{pit} \in \{0, 1\}, \qquad \forall t \in T, \forall p \in P, \ \forall i \in \{1, \ldots, N_M\} \tag{10}$$

$$t_m \in \mathbb{Z} \tag{11}$$

where $(\alpha_1 + \alpha_2) = 1$.

In the previously presented model, denoted as CC-IP, the objective function (1) seeks both the minimization of costs and total execution time. The weights α_1 and α_2 define the relevance of each objective in the opinion of users. This allows for a fine tuning of the model, *i.e.* users can choose if they want a "fast" execution or a "low-budget" execution. Generally, since these objectives are conflicting, both objectives cannot arrive at optimal levels simultaneously. For example, if the value of α_1 is close to 1 the model prioritizes "low-budget" solutions with high execution time. Constraint (2) ensures that the financial costs do not exceed the maximum financial cost set by users. Inequalities (3) and (4) enforce that the storage and memory capacity are, respectively, sufficiently large to meet the requirements in each period. Similarly, constraint (5) guarantees that the processing power is at least large enough to satisfy the total demand. Inequalities (6) ensures that the number of selected virtual machines does not exceed the cloud providers limit while constraint (7) together with objective function guarantee the correct interpretation of variable t_m. Constraint (8) enforces that there will be no gaps of time in feasible solutions (*i.e.* if a virtual machine is instantiated at time $t + 1$, then it will also be selected at time t). Note that all virtual machine combinations yield feasible solutions in CC-IP, thus the constraints (9) were added to establish an order between the virtual machines and eliminate symmetry. Constraints (10) and (11) define the binary and integrality requirements on the variables.

The CC-IP model was initially designed for general tasks (standalone applications). However it was easily adapted to be applied in scientific workflows that execute using the *First Activity First* (*i.e.* FAF) dataflow strategy, since in FAF, all CAs associated to a specific activity are independent. In addition, all CAs of a subsequent activity in the flow are only executed when the CAs associated to the previous activity finish, *i.e.* all CAs for an activity are treated as a bag-of-task application. However, the entire workflow is not considered as a bag-of-tasks application since there is data dependency among the activities and CAs. This way, we can dimension the cloud environment before each activity by running CC-IP. However, this approach is not suitable when users run workflows using the FTF model, since CAs from different activities may be executing at the same moment. Fig. 7 presents how we invoke CC-IP in a scientific workflow scenario. In Fig. 7

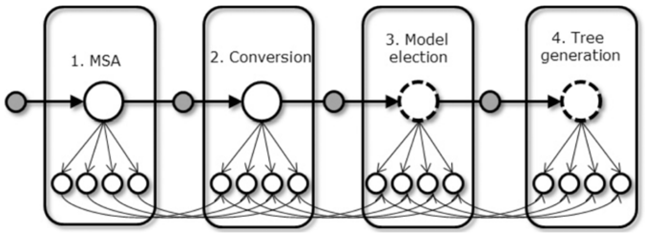


Fig. 7. Coupling CC-IP and workflow execution [77].

each big white circle is an activity that is composed by tasks (small white circles), called here CAs. Before the start of the execution of each activity, CC-IP is invoked (gray circles) to estimate the amount of VMs needed for the execution of the next activity. This way, CC-IP can consider CAs associated to the same activity as independent sets.

5.2. GraspCC approach

Exact procedures and brute-force search have often proved incapable of finding solutions, as they are extremely time-consuming, particularly for real-world problems as large-scale scientific workflows. In the context of this article, the time needed to allocate the necessary virtual machines has to be negligible when compared to the total execution time of the workflow. The effective use of heuristics and metaheuristics provides sub-optimal solutions in a reasonable time. Actually, in the scientific workflow domain, these sub-optimal solutions are acceptable since engines such as SciCumulus are able to adapt the amount of virtual machines on the execution course of the workflow. In this way, if an approach can estimate an optimal or sub-optimal set of virtual machines to allocate it can speed up the performance of the parallel workflow execution.

In this context, we have designed and implemented a Greedy Randomized Adaptive Search Procedure (GRASP) which has been successfully applied in many combinatorial optimization problems [74–76]. Although the heuristic proposed in our previous work [28], named GraspCC, was initially designed for standalone applications, previously experiments with different workflows [77] showed its viability for scientific workflow scenarios. GraspCC is composed of two phases: a construction phase *coCC* and a local search phase *lsCC*. The construction phase finds an initial solution that may be later improved by the local search phase.

In order to describe this method, we need some additional notation. We define a *solution* $\{(p_1, i_1, t_1), (p_2, i_2, t_2), \ldots\}$ as a set of 3-tuples (p, i, t) representing that virtual machine i of type p was allocated on period t. Let S be the set of all possible solutions. A solution $s \in \mathbb{S}$ is said to be *feasible* if s respects the user's requirements. We also denote $t_m(s) = \max_{(p,i,t)\in s} t$ as the last period that a virtual machine was selected. Moreover, we define below a *cost* *function* $F : \mathbb{S} \to \mathbb{R}$, which measures the quality of the solution. Similar to the objective function (1), the function F tries to minimize financial costs and total execution time (12) while relaxing the inequality (13), penalizing the infeasibility regarding maximum time and financial cost. The parameters λ_1 and λ_2 are penalty coefficients associated with the violation of time and cost requirements, respectively.

$$F(s) = \left(\alpha_1 \sum_{(p,i,t)\in s} c_p + \alpha_2 t_m(s)\right) \tag{12}$$

$$+ \lambda_1(\max\{0, t_m(s) - T_M\}) + \lambda_2 \left(\max\left\{0, \sum_{(p,i,t)\in s} c_p - C_M\right\}\right). \tag{13}$$

Algorithm 1: *GraspCC*

```
1  Input: $P, C_M, T_M, D_S, M_C, G_f, \alpha_1, \alpha_2, \lambda_1, \lambda_2$
2  Output: solution $s^*$;
3  $s^* := \emptyset;\ F(s^*) := \infty;\ i := 0;$
4  while $i \leqslant iter$
5      $s := coCC(P, C_M, T_M, D_S, M_C, G_f, \alpha_1, \alpha_2, \lambda_1, \lambda_2);$
6      $s := lsCC(s, P, C_M, T_M, D_S, M_C, G_f, \alpha_1, \alpha_2, \lambda_1, \lambda_2);$
7      if $(F(s) < F(s^*))$ and ($s$ is feasible)
8          $s^* := s;\ i := 0;$
9      end if
10     $i := i + 1;$
11 end while
12 return $s^*$;
```

We present GraspCC algorithm in Algorithm 1, where the parameter *iter* denotes the maximum number of iterations without improvement in the best solution found. The first task in every iteration of the GraspCC is to construct a solution starting from an empty set, in a greedy randomized fashion. This task is performed by algorithm *coCC* which is described in Algorithm 2. In this method, the ordered set L_P is defined, in line (3), as the set of virtual machines $p \in P$ such that its elements appear in descending order of financial cost and processing power $(\alpha_1 c_p + \alpha_2 g_p)$. At each iteration in lines (4)–(7) we choose a virtual machine $\overline{p}$ among the first β virtual machines to be added to the first period of time. Note that the parameter β defines the degree of randomness the construction phase will have. This process is repeated until the solution satisfies the disk and memory requirements (for $t = 1$), however the maximum time and financial cost constraints are relaxed in order to achieve diversity in the initial solution. Moreover, in lines (9)–(12) the chosen virtual machines are replicated in future periods until the processing demand is satisfied.

There is no guarantee that the construction method returns a feasible or a locally optimal solution with respect to some neighborhood. Therefore, the solution s may be improved by a local search procedure denominated *lsCC* (Algorithm 3). The neighborhood $N_r(s)$ is defined as the family of all solutions obtained by exchanging r tuples in s with another r tuples not in s. These exchanges are made extensively with the strategy of the first improvement. The *lsCC* method starts with the solution provided by the construction phase. It iteratively replaces the current solution by that with minimum cost function F within its neighborhood (lines (3)–(9), where "s improving" means that there is some solution in the neighborhood of the current solution with cost function better than the current s). The local search halts when no better solution is achieved in the neighborhood of the current solution. In this article we employed a neighborhood with $r \leq 2$, since for values of $r > 2$ it is computing-intensive to perform an exhaustive search.

Algorithm 2: *coCC*

```
1  Input: $P, C_M, T_M, D_S, M_C, G_f, \alpha_1, \alpha_2, \lambda_1, \lambda_2$
2  Output: solution $s$;
3  $s := \emptyset;\ L_P := Order(P);$
4  while $(\sum_{p|(p,i,1)\in s} d_p < D_S)$ or $(\sum_{p|(p,i,1)\in s} m_p < M_C)$
5      Choose virtual machine $\overline{p}$ (index $\overline{i}$) randomly among the
       first $\beta$ elements of $L_P$
6      $s := s \cup \{(\overline{p}, \overline{i}, 1)\};$
7  end while
8  $\overline{t} := 2;$
9  while $(\sum_{p|(p,i,t)\in s} g_p < G_f)$
10     $s := \bigcup_{(p,i,1)\in s}(p, i, \overline{t}) \cup s;$
11     $\overline{t} := \overline{t} + 1;$
12 end while
13 return $s$
```

Algorithm 3: *lsCC*

```
1  Input: $s, P, C_M, T_M, D_S, M_C, G_f, \alpha_1, \alpha_2, \lambda_1, \lambda_2$
2  Output: solution $s$;
3  while ($s$ improving)
4      for all $\overline{s} \in (N_1(s) \cup N_2(s))$
5          if $F(\overline{s}) < F(s)$
6              $s := \overline{s}$
7          end if
8      end for
9  end while
```

5.3. Genetic algorithm

A Genetic Algorithm (or simply GA) is an optimization technique for solving complex problems [25–27]. In a GA, a set of candidate solutions to an optimization problem is evolved in order to find better solutions. Each candidate solution is associated with a set of chromosomes (*i.e.* set of properties) that are mutated. In general, the solutions are represented as strings of 0 and 1, but other implementations are also possible [78]. A GA is then composed by a genetic representation of the solution domain and a fitness function to evaluate the candidate solutions. In addition, a multi-objective GA takes into account several different goals, which may be conflicting, to obtain a problem solution as aforementioned.

GA are relatively simple and fast algorithms [26], besides being indicated for cases where the search space is large and the problem does not require the optimal solution, which is the case of the prediction of the amount of virtual machine to be allocated for a cloud-based parallel scientific workflow execution.

Oliveira et al. [24] proposed a service named SciDim that was coupled to SciCumulus engine to estimate the amount of virtual machines to allocate for a parallel workflow execution. SciDim was implemented using JGAP framework [79], which provides basic mechanisms that can be easily used to apply evolutionary principles to the problem to be solved. In the implementation presented by Oliveira et al. [24] each chromosome is composed by four genes, each one associated to a specific type of virtual machine available to use in the workflow executions (in [24] the authors follow Amazon EC2 model and they only consider micro, large, extra-large and extra-large high capacity virtual machine types). In the implementation proposed in [24] each allele represents the amount of virtual machines of a specific type to be instantiated. Users have to specify a lower and an upper limit, which define the values for each type of virtual machine. This limit is quite similar to the inequality (6) in the CC-IP model. The used fitness function is based on a multi-objective cost function proposed

Table 2
Comparison of *GraspCC* Metaheuristic and SciDim service with $\alpha_1 = 0.5$ and $\alpha_2 = 0.5$.

Instances	GraspCC				SciDim			
	Function cost	Solution value		Total time (s)	Function cost	Solution value		Total time (s)
		Time	Financial			Time	Financial	
modgen-real_am_p	0.0271	1	2.88	0.05	0.0271	1	2.88	3.12
raxml-real_am_p	0.0271	1	2.88	0.05	0.0271	1	2.88	3.14
modgen-real_am_c	0.0230	1	2.04	0.12	0.0240	1	3.04	4.61
raxml-real_am_c	0.0230	1	2.04	0.12	0.0235	1	2.56	4.60

by Oliveira et al. [59] where the authors considered all costs involved in the parallel execution of the workflow as follows. The workflow execution is based on three objectives: total execution time, financial cost and reliability (for the experiments performed in this article we did not consider the reliability objective since the GraspCC does not take availability into account). The authors use a weighted cost function to set the weight for each objective. For more information about the multi-objective function please refer to Oliveira et al. [59]. For each VM_j in the set VM that is idle and requests for a activation, it is then performed a search for the best CA ca_i in the list of available CAs (named Ca') to execute in VM_j following the 3-Objective cost function. Given $VM_j \in VM$ as the next virtual machine to execute a CA, the optimizer has to find $ca_i \in Ca'$ which minimizes the following cost function:

$$f(ca_i, VM_j) = \alpha_1 \times P(ca_i, VM_j) + \alpha_2 \times R(ca_i, VM_j) + \alpha_3 \times M(\varphi). \quad (14)$$

By allowing users to inform the three alpha values, it is possible to fine tune priorities in the workflow execution parameters. Note that users do not estimate the optimal alpha values. Actually, they perform a fine tuning during several workflow executions. After testing a varied amount of values for each alpha, they can configure the best value for their experiment. This cost function is used in the genetic algorithm of SciDim to define a set of virtual machines near to ideal for the execution of a scientific workflow in the cloud. This way, the fitness function is defined as:

$$g(x) = 1 - \left\lceil \frac{\left(\sum_{i=1}^{n} f(ca_i, VM_j)\right)}{|CA|} \right\rceil. \quad (15)$$

The value of $g(x)$ has to be compared with the constraints set by the user for total execution time and maximum financial cost. If the generated value complies with these constraints it can be considered for evaluation. Note that this value is always smaller than 1 and the algorithm chooses the configuration that provides the higher value and complies with users' constraints. During the process of evolution, chromosomes are exposed to various operators such as mutation and then are selected for the next generation based on the fitness value. The fitness function was developed considering that all objectives of the cost function are taken into account. In each generation, the authors performed a simulation of the entire workflow execution based on a simulator [80] that implements the cost function. For the experiments presented by Oliveira et al. [24], the authors use standard values widely used in similar problems in the literature. The initial population of chromosomes is generated randomly by a JGAP method: 100 individuals. JGAP was configured to perform 200 generations since we cannot consume a long time in the environment configuration instead of using this time for workflow execution. Besides they set to have a crossing rate of 35% of the population and an 8% rate of mutation of individuals, which are default values for JGAP. In the following we present the scientific workflow used as case study for both GraspCC and SciDim and the comparison between these approaches.

6. Experimental evaluation: GraspCC and SciDim

GraspCC and SciDim were re-implemented in ANSI C in order to provide a fair comparison between these approaches. All simulated experiments were performed on a computer with processor Intel Core i7 3.4 Hz and 12 GB of RAM under Linux (Ubuntu12.04) operating system. The algorithms were tested over input dataset containing historical execution information of ModelGenerator and RAxML that are part of SciPhylomics and virtual machine types of Amazon EC2 cloud (for more details about the requirements of ModelGenerator and RAxML, see Table 5). All information necessary to both GraspCC and SciDim were extracted from the SciCumulus provenance repository [59] by executing the following simple query:

```
select ac.starttime, ac.endtime, ac.workspace
from hworkflow w, hactivity a, hactivation ac
where w.wkfid = a.wkfid
and a.actid = ac.actid
and w.wkfid = % Workflow ID %
and ac.status = 'FINISHED'
order by ac.starttime
```

The experiments, reported in Table 2, are a comparison of the GraspCC against genetic algorithm implemented in SciDim, in terms of solution quality and execution time. The used parameter values and execution number of GraspCC and SciDim were the same of previous work of Coutinho et al. [28] and Oliveira et al. [24], respectively. Both objectives of the cost function were normalized as done by [28]. Each instance (*i.e.* set of historical execution information) is identified by the label *n_c_g*, where *n* is the application name, *c* is the cloud provider (*i.e.* Amazon EC2) and *g* is the generation of Amazon's virtual machines of general use (*i.e.* *p* or *c* for previous and current generations, respectively). The first column is the instance name. The next three columns present the best solution found by GraspCC and the costs (not normalized) that compose it: total execution time and financial cost, respectively. The fifth column shows the execution time in seconds for GraspCC to find the best solution. Analogously, the last four columns present the best solution found by SciDim, the costs that compose it (the execution time and financial costs), and the time in seconds spent to find the best possible solution. All feasible solutions found by SciDim were worse than the ones provided by GraspCC solution, even spending more time to find it.

Although the theoretical results produce strong indications that GraspCC approach is more adequate than genetic algorithms, we need to effectively compare the two approaches using real executions of the workflow. This way, to compare the produced solutions by GraspCC and SciDim we executed the fragment of SciPhylomics (presented in dashed red line in Fig. 4) using SciCumulus engine in Amazon EC2 cloud. Recently, sensing the customers' need for more computing power in smaller and cheaper virtual machines, Amazon has added a number of new virtual machines in its Elastic Cloud Compute (EC2) service called next generation virtual machines. In this experiment we considered virtual machines from both previous and current generations. The configurations of these virtual machines are described in Table 3,

Table 3
Configuration of the Amazon EC2 virtual machines.

Generation	Virtual machine	RAM (GB)	Disk storage (GB)	GFLOP number	Financial cost ($)
Previous	m1.small	1.7	160	19.2	0.06
	m1.medium	3.75	410	38.4	0.12
	m1.large	7.5	840	76.8	0.24
	m1.xlarge	15	1680	153.6	0.48
Current	m3.xlarge	15	0	249.6	0.50
	m3.2xlarge	30	0	499.2	1.00

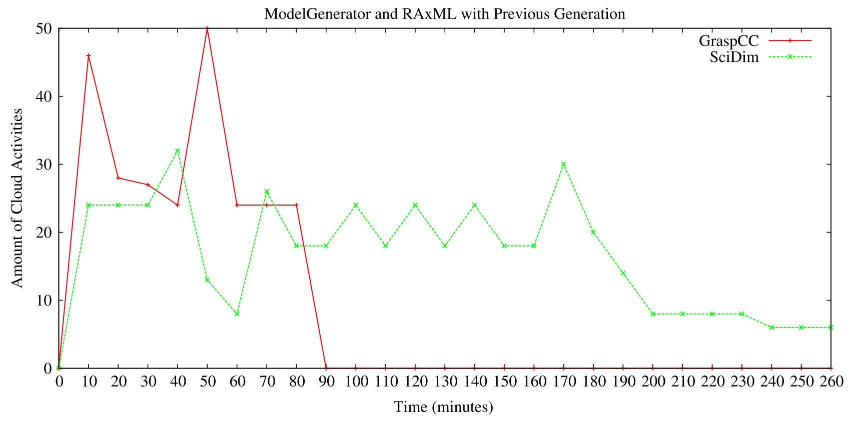


Fig. 8. Using virtual machines of the previous generation of Amazon EC2.

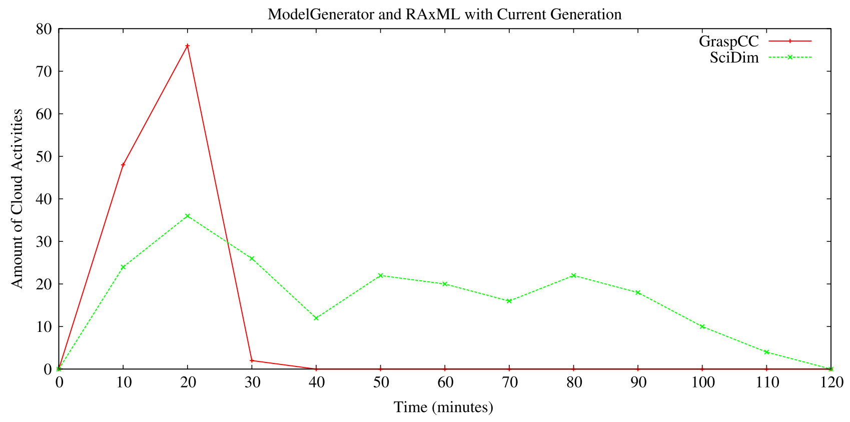


Fig. 9. Using virtual machines of the current generation of Amazon EC2.

where the columns represent the virtual machine generation of Amazon EC2, the virtual machine name, the requirements of RAM memory, disk storage, gflop number and the financial cost to allocate this virtual machine per hour, respectively. In the first part of this experiment, we considered only the virtual machines of the previous generation of Amazon EC2. For this, we allocated the following suggested amount of resources for each approach: (i) GraspCC—6 m1.xlarge virtual machines for 1 h and (ii) SciDim—6 m1.small, 3 m1.medium, 3 m1.large, 3 m1.xlarge virtual machines for 1 h, for both applications (ModelGenerator and RAxML). Next, we considered the virtual machines of both previous and current generations, allocating the following resources for each approach: (i) GraspCC—3 m3.xlarge, 1 m1.xlarge, 1 m1.small virtual machines over 1 h for both applications (ModelGenerator and RAxML) and (ii) SciDim—6 m1.small, 4 m1.medium, 5 m1.large, 2 m3.large virtual machines per 1 h for ModelGenerator, and 4 m1.small, 3 m1.medium, 2 m1.large, 1 m1.xlarge, 1 m3.2large per 1 h for RAxML. Each allocated virtual machine is based on a 64-bit Linux Cent OS 5.5. The SciPhylomics workflow consumed a dataset of 200 reduced multi-fasta files containing protein sequences.

Figs. 8 and 9 show the execution of the workflow fragment (ModelGenerator and RAxML) in the Amazon EC2 Cloud using the virtual machine configurations given by GraspCC and SciDim solutions. The graphics present the amount of CAs (axis Y) executing in a given period of time (X axis). In Fig. 8, we observe the solutions from GraspCC and SciDim using only Amazon's previous generation virtual machines of general use (instances *modgen-realp* and *raxml-realp* of Table 2). In Fig. 9, we see the solutions from GraspCC and SciDim using virtual machines of both Amazon's previous and current generations of general use (instances *modgen-realc* and *raxml-realc* of Table 2). By analyzing the results presented in Fig. 9 we can state that GraspCC suggested that we needed 5 virtual machines for 1 h with financial cost of $2.04 for each application, totalizing 2 h and a financial cost of $4.08 for the execution of the workflow fragment, but the real execution lasted less 1 h (100% less) with an approximated financial cost of $2.04.

Note that the execution of the workflow fragment using the amount of virtual machines following the allocation given by GraspCC was about 64% smaller than execution that followed the configuration provided by SciDim for the previous generation virtual machines. In addition the execution of the workflow fragment with new generation virtual machines was almost 69% smaller using GraspCC allocation in comparison with SciDim's. Although the amount of cores and ECU (computing unit in Amazon EC2) are the same for both configurations, this behavior can be explained since GraspCC suggested a reduced set of more powerful virtual machines to instantiate while SciDim suggested using a larger set of less powerful virtual machines. This way, using more powerful virtual machines we benefit from the fact that Amazon virtual machines are optimized for general use. On the other hand, when we are using more virtual machines the communication overhead impacts since messages have to be sent and received for managing the workflow execution. Thus, the execution performance is severely affected when we have several virtual machines. This way the configuration provided by GraspCC was more adequate than SciDim configuration. This can be stated by analyzing Fig. 8 where the amount of CAs per period of time is considerably higher in most occasions. Thus, the obtained results provide a strong indication that the GraspCC offers the adequate solution for determining the amount of virtual machines to instantiate in a federated cloud for a parallel workflow execution. The next section presents the proposed solution for federated clouds.

7. Allocating virtual machines for parallel execution of scientific workflows in federated clouds

A Federated Cloud is composed by multiple cloud computing providers, public or private, that can work together in order to meet the QoS requirements of cloud consumers [15,16], or in the context of this article the restrictions imposed by users. According to Buyya et al. [15], cloud providers have established data centers in multiple geographical locations to provide redundancy and reliability in failure cases to a large number of consumers (*e.g.* users) from around the world. For example, Amazon is hosted in multiple locations world-wide, like US, Japan, Europe and Brazil as presented in Fig. 10. Especially for users that execute their workflows in parallel in clouds, the concept of federated clouds is interesting in three main scenarios (this is a non-exhaustive list). The first one is related to the maximum number of virtual machines that can be allocated in a provider. It is quite probable that a specific workflow execution may need more virtual machines than the maximum allowed by the cloud provider (to meet users' requirements), thus virtual machines have to be allocated in several providers. The second scenario is to avoid costly data transfers. In some experiments, such as bioinformatics ones, DNA sequences have to be compared to the entire NCBI database for example. However, this is a 15 GB database that cannot be easily transferred. This way, it is interesting to execute some activities of the workflow (*i.e.* the ones that query the database) geographically "near" the database in a specific provider and then transfer only the partial (and small) generated results of the next activities in the flow. Finally the third scenario is when the used provider cannot offer new requested virtual machines and then rents resources in a second provider. However, to use existing SWfMS that execute in clouds, users must express their preference about the provider and the location where they desire to execute their workflows and the cloud provider does not offer native and automatic tools for scaling their resources across multiple geographically distributed locations. Moreover, since users can use several different providers, the amount of virtual machine types to allocate grows exponentially. So, it is far from trivial for users to determine the types of virtual machines to use, the amount of each type to allocate and in which provider and location to allocate these virtual machines to execute their parallel workflows. Currently, cloud providers are not able to meet users' requirements regards a parallel workflow execution since they are not aware of the workflow execution. In fact, the responsibility of managing the workflow execution and meeting the users' requirements are of the SWfMS. To the best of our knowledge there is a few SWfMS that

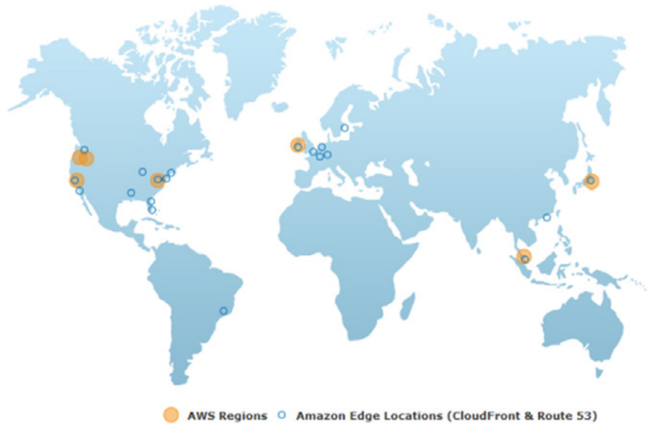


Fig. 10. Global infrastructure of Amazon web service [17].

support the parallel execution of scientific workflows in federated clouds. However, none of then estimates the amount of virtual machines to allocate for this type of environment.

In addition, currently commercial cloud providers do not implement mechanisms for cloud federation yet. In fact, when users need to allocate virtual machines from different providers they commonly apply in house (and commonly not optimized) solutions to perform communication of processes in data centers of distinct cloud providers. To overcome this limitation of the cloud providers, in the context of this article, we adapted SciCumulus engine to execute in a federation of clouds by creating a shared storage that is accessed by all virtual machines involved in the workflow execution. This shared storage was created in Amazon S3 and it is shared using S3fs mechanism [81]. S3fs is a file system that allows for users to mount in a bucket in Amazon S3 environment a local file system. It stores files natively and transparently in Amazon S3, allowing all virtual machines to map a local directory to a bucket. This way, all virtual machines can access the same files during workflow execution. This solution introduces several issues that have to be considered when we are estimating the amount of virtual machines to allocate. These issues include communication overhead introduced by synchronization of files in S3 and the synchronization messages sent between all virtual machines of different providers.

Thus, in order to take into account the aforementioned issues in the virtual machine allocation for the parallel execution of workflows in federated clouds problem, we extended the previously presented model to consider new issues such as communication between providers. To describe this new scenario we extend the previous notation as presented next.

Let q be the number of cloud providers, P_j be the set of virtual machine types offered by the cloud provider j, $1 \leq j \leq q$, and $P = P_1 \cup P_2 \cup \cdots \cup P_q$ the set of all virtual machines from different providers. Now, each virtual machine type also has another associated cost $\overrightarrow{c}_{pip'i'}$, that is the communication cost from the virtual machine i of type p to another virtual machine i' of type p'. We consider that $\overrightarrow{c}_{pip'i'} = (uc_p + dc_{p'} + sc) * s$, if p and p' belong to distinct cloud providers, or $\overrightarrow{c}_{pip'i'} = cs_p * size\_data$, otherwise, where uc_p is the upload cost from a machine of the type p, $dc_{p'}$ is the download cost to a machine of the type p', sc is the storage cost of the transmitted data, cs_p is the communication cost of virtual machine type p with others virtual machine types of the same cloud provider and s is the average size of the transmitted data. Moreover, let N_M^j be the maximum limit of allocated virtual machines by the user in each cloud provider j in each period of time, where $\Pr(p)$ indicates the provider index of virtual machine $p \in P$. We also define other two binary variables: y_{pi} for each $p \in P$, $i \in \{1, \ldots, N_M^{\Pr(p)}\}$,

Table 4
Notations for federated clouds environment.

Notation	Description
P_j	The set of virtual machines types offered by cloud provider j
P	The set generated by the union of all sets P_j $P = \{P_1 \cup P_2 \cup \cdots \cup P_q\}$
N_M^j	The maximum number of allocated virtual machines by the user in each cloud provider j
$\Pr(p)$	The cloud provider index of virtual machine $p \in P$
$\overrightarrow{c}_{pip'i'}$	The communication cost from a virtual machine i of type p to another virtual machine i' of type p'
uc_p	The upload cost from a virtual machine p
dc_p	The download cost to a virtual machine p
sc	The storage cost of the transmitted data
s	The average size of the transmitted data
cs_p	The communication cost of virtual machine type p with others virtual machine types of the same cloud provider
y_{pi}	$y_{pi} = 1$, if and only if virtual machine i of type p is allocated in some period of time; otherwise, $y_{pi} = 0$
$\overrightarrow{z}_{pip'i'}$	$\overrightarrow{z}_{pip'i'} = 1$, if and only if $y_{pi} * y_{p'i'} = 1$; otherwise, $\overrightarrow{z}_{pip'i'} = 0$

such that $y_{pi} = 1$ if and only if virtual machine i of type p is purchased; or $y_{pi} = 0$, otherwise; and $\overrightarrow{z}_{pip'i'}$ for each $p, p' \in P$, $i \in \{1, \ldots, N_M^{\Pr(p)}\}$, $i' \in \{1, \ldots, N_M^{\Pr(p')}\}$, such that $\overrightarrow{z}_{pip'i'} = 1$ if and only if $y_{pi} * y_{p'i'} = 1$; or $\overrightarrow{z}_{pip'i'} = 0$, on the other case. These last two variables allow that the communication cost between two allocated virtual machines is considered. Tables 1 and 4 present the used notation for the mathematical formulation.

Thus, to be able to estimate the amount of virtual machines to use for a specific workflow execution in federated clouds we formulated the problem as an integer programming problem by adding this part:

$\sum_{p\in P} \sum_{p'\in P} \sum_{i=1}^{N_M^{\Pr(p)}} \sum_{i'=1}^{N_M^{\Pr(p')}} \overrightarrow{c}_{pip'i'} \overrightarrow{z}_{pip'i'}$ to the objective function (1), removing the constraint (6) and adding some constraints to the formulation of Section 5.1. The new objective function and the additional constraints are described as follows:

$$\text{(CC-IP-fed)} \quad \min \left(\alpha_1 \left(\sum_{p\in P} \sum_{i=1}^{N_M^{\Pr(p)}} \sum_{t\in T} \overrightarrow{c}_p x_{pit} + \sum_{p\in P} \sum_{p'\in P} \sum_{i=1}^{N_M^{\Pr(p)}} \sum_{i'=1}^{N_M^{\Pr(p')}} \overrightarrow{c}_{pip'i'} \overrightarrow{z}_{pip'i'} \right) + \alpha_2 t_m \right) \quad (16)$$

$$\sum_{p\in P_j} \sum_{i=1}^{N_M^{\Pr(p)}} x_{pit} \le N_M^{\Pr(p)}, \quad \forall j = 1 \ldots q, \forall t \in T \quad (17)$$

$$y_{pi} \ge \overrightarrow{z}_{pip'i'}, \quad \forall p, p' \in P, \forall i \in \{1, \ldots, N_M^{\Pr(p)}\}, i' \in \{1, \ldots, N_M^{\Pr(p')}\} \quad (18)$$

$$y_{p'i'} \ge \overrightarrow{z}_{pip'i'}, \quad \forall p, p' \in P, \forall i \in \{1, \ldots, N_M^{\Pr(p)}\}, i' \in \{1, \ldots, N_M^{\Pr(p')}\} \quad (19)$$

$$y_{pi} + y_{p'i'} - 1 \le \overrightarrow{z}_{pip'i'}, \quad \forall p, p' \in P, \forall i \in \{1, \ldots, N_M^{\Pr(p)}\}, i' \in \{1, \ldots, N_M^{\Pr(p')}\} \quad (20)$$

$$\sum_{t\in T} x_{pit} \le y_{pi}|T|, \quad \forall p \in P, \forall i \in \{1, \ldots, N_M^{\Pr(p)}\} \quad (21)$$

$$y_{pi} \in \{0, 1\}, \quad \forall p \in P, \forall i \in \{1, \ldots, N_M^{\Pr(p)}\} \quad (22)$$

$$\overrightarrow{z}_{pip'i'} \in \{0, 1\}, \quad \forall p, p' \in P, \forall i \in \{1, \ldots, N_M^{\Pr(p)}\}, i' \in \{1, \ldots, N_M^{\Pr(p')}\} \quad (23)$$

where $(\alpha_1 + \alpha_2) = 1$.

In the above model, denoted as CC-IP-fed, the objective function (16) also seeks both the minimization of financial costs and total purchased time. The inequalities (17) ensure that the number of selected virtual machines does not exceed the limit imposed by each cloud provider. The constraints (18)–(21) ensure that the communication cost between two allocated virtual machines is considered. The inequalities (18)–(20) are the linearization classical restrictions of the multiplication of variables y while the constraint (21) binds the variables x and y, i.e., if $x_{pit} = 1$ for some t, this means that $y_{pi} = 1$. The constraints (22) and (23) define the binary and integrality requirements on the variables.

As GraspCC presented better performance results in the single-provider cloud when compared to the genetic algorithm of SciDim, we also extended GraspCC to be used in the context of federated cloud environments. It was not necessary to modify the *GraspCC* skeleton, only the communication cost, defined as $\alpha_1 \sum_{(p,i,t)\in s} \sum_{(p',i',t')\in s} c_{pip'i'}$, was added to the function F, generating a new function F':

$$F'(s) = \left(\alpha_1 \left(\sum_{(p,i,t)\in s} c_p + \sum_{(p,i,t)\in s} \sum_{(p',i',t')\in s} c_{pip'i'} \right) + \alpha_2 t_m(s) \right) \quad (24)$$
$$+ \lambda_1(\max\{0, t_m(s) - T_M\}) + \lambda_2 \left(\max\left\{0, \sum_{(p,i,t)\in s} c_p - C_M \right\} \right). \quad (25)$$

Based on the previously presented GraspCC-fed we performed simulated and real experiments using SciPhylomics and SciCumulus engine in a federated environment that is composed by virtual machines from Amazon EC2 and Microsoft Azure. These experiments are presented in the next section.

8. Experimental results for federated clouds

In this section, we first compare the GraspCC-fed with the CC-IP-fed in terms of solution quality and execution time through simulated experiments. Remark that GraspCC-fed yields heuristic solutions while CC-IP-fed provides exact solutions. Next, we also evaluate the GraspCC-fed in a real parallel execution of a workflow using the bioinformatics workflow SciPhylomics for phylogenomic analysis [30] and adapted SciCumulus engine that we called SciCumulus-fed. Thus, the GraspCC-fed was executed using the same workflow fragment (also used in previous tests) for federated clouds environments. The obtained results for simulated and real tests are presented as follows.

8.1. Comparing GraspCC-fed and CC-IP-fed

The proposed GraspCC-fed and CC-IP-fed algorithms were implemented in ANSI C and the CPLEX 12.4 [82] was used to solve

Table 5
Instance description.

Instance	RAM (GB)	Disk storage (GB)	GFLOP	Time (h)	Cost ($)	Data size (MB)	Cloud providers
nug22-sbb	77	51	5 067 533	12	343	103.01	Amazon, Google
nug24-sbb	154	103	14 741 914	48	998	192.93	Amazon, Google
nug25-sbb	214	142	28 792 800	60	1950	262.17	Amazon, Google
nug28-sbb	528	352	67 720 666	72	4586	174.87	Amazon, Google
nug30-sbb	918	612	120 929 760	84	8190	95.61	Amazon, Google
cms-1000	1500	20	216 000 000	24	1728	0.0	Amazon, Google
cms-1250	1875	25	270 000 000	24	2304	0.0	Amazon, Google
cms-1500	2250	30	324 000 000	24	2592	0.0	Amazon, Google
mod-gen	4	2	3 317 760	24	100	6.95	Amazon, Azure
raxml	2.5	2	3 317 760	24	100	6.95	Amazon, Azure

the CC-IP-fed formulation. The algorithms were tested over input dataset containing historical execution information of real applications and virtual machine types available in commercial clouds. In addition to ModelGenerator and RAxML, the requirement information of the other two HPC applications were also used. We have chosen to consider provenance information from different applications to diminish the bias. Actually, this information helped us to calibrate the model. One is a branch-and-bound algorithm for solving the Quadratic Assignment Problem (QAP) [83]. The QAP has the objective of finding a minimum cost allocation of facilities to locations, where the cost is obtained by the sum of products *distance* $*$ *flow*. So the distance and flow matrices are considered in this problem. Those instances were constructed by using the execution requirements of the *nug* class from QAPLIB [84] with 22, 24, 25, 28 and 30 facilities. Another application is a typical analysis user job for the CMS experiment [85] run at the LHC/CERN grid, by using the CRAB software. The considered numbers of tasks per jobs are 1000, 1250 and 1500 in each instance, respectively. Finally, we also used virtual machine configurations of three commercial clouds Amazon EC2, Google Cloud Platform and Microsoft Azure to compose those instances for the simulated tests. However, due to different characteristics of the applications, we used only the HPC virtual machine configurations of the Amazon EC2 and the Google Cloud Platform with the QAP algorithm and the CMS experiment, and the virtual machine configurations of the general use type of Amazon EC2 and the default type of the Microsoft Azure with ModelGenerator and RAxML. The requirements of the applications and the cloud providers considered in those instances are described in Table 5. The columns represent the instance name, the informed restrictions of RAM memory, disk storage, gflop number, maximum time execution and maximum financial cost, the average data size of each execution of the application and the used cloud providers, respectively. The maximum number of virtual machines allowed in each cloud provider was 20. These simulated experiments were performed in the same environment of the previous tests.

Experimental results presented in Table 6 are a comparison of the GraspCC-fed with the CC-IP-fed, in terms of solution quality and execution time. The used parameters were: $\lambda_1 = 1000$ and $\lambda_2 = 1000$, $\beta = \max_{p \in P}(N_M^j)$ and $iter = 40$. The used values of the parameters α_1 and α_2 represent the extreme cases where variables are removed from the objective function ($\alpha_1 = 0$ or $\alpha_2 = 0$) and the case in which all variables are considered ($\alpha_1 \neq 0$ and $\alpha_2 \neq 0$). The maximum number of virtual machines N_M^j in each provider j is the same as stated in the provider Amazon EC2.[2] The values of other parameters β, λ_1, λ_2 and *iter* were obtained empirically as it is usually done in the parameter settings of metaheuristics. Both objectives of the cost function were normalized due to their distinct range values. The normalization procedure updates the objective values so that they share the same minimum and maximum values, 0 and 1, respectively. Thus, the financial cost was divided by the sum of two components. The first one is the product of the most expensive virtual machine financial cost, the maximum number of virtual machines and the maximum time informed by the cloud consumer. The second component of the sum is the product of the average size of the sent data and the average communication cost between the virtual machines of all cloud providers. Similarly, the execution time component of the objective function was divided by the maximum time informed by users.

In Table 6, the first column is identified by the application name. The next four columns present the best feasible solution found by CC-IP-fed and the costs (not normalized) that compose it: financial costs for virtual machine hiring and communication, and its execution time, respectively. The sixth column shows the execution time in seconds for CC-IP-fed to solve the problem. Analogously, the next five columns represent the same things for the best solution found by GraspCC-fed. Finally, the last two columns show the used values for α_1 and α_2 in the cost function.

For some instances, the CC-IP-fed took a long time to prove the optimality of the solution, therewith we consider in our experiments a maximum execution time restriction of 24 h. The exact method was not able to find the optimal solution in instances marked with (*) in Table 6 within this time limit. Note that GraspCC-fed presented an improvement of the execution time, in average 99.35% less than the execution time of CC-IP-fed. Furthermore, the GraspCC-fed heuristic found a better or equal solution than the CC-IP-fed formulation in most instances (which are highlighted in Table 6), presenting a percentage difference from the best feasible solution found by CC-IP-fed of 5.43%, in average.

Despite the GraspCC-fed does not find a better solution than the CC-IP-fed for some instances, its execution time is always smaller than the one presented by the CC-IP-fed. For example, GraspCC-fed found a solution 46.67% far from the formulation for the instance *cms-1000* with $\alpha_1 = 0$ and $\alpha_2 = 1$. However, GraspCC-fed took much less time to obtain this solution, only 57 s against 7.7 h of the formulation. If this execution time was added to the solution time of application, the total time using GraspCC-fed and CC-IP-fed would be 4.02 and 10.7 h, respectively. Thus, even when GraspCC-fed does not find the best solution, it can be considered an attractive alternative to solve the virtual machine allocation problem in federated clouds.

8.2. Executing federated cloud scenarios in real commercial clouds

In this subsection we present an evaluation of the configuration provided by GraspCC-fed using a real experiment executed in a federated cloud scenario. In order to study the feasibility of this approach in federated cloud scenarios, we use again the bioinformatics workflow SciPhylomics. Thus, the main goal of this

2 https://aws.amazon.com/ec2/faqs/.

Table 6
Results of GraspCC-fed metaheuristic and CC-IP-fed mathematical formulation using CPLEX.

Instances	CC-IP-fed					GraspCC-fed					α_1	α_2
	Function cost	Solution value			Total time (s)	Function cost	Solution Value			Total time (s)		
		Financial costs		Time			Financial costs		Time			
		Hiring	Communication				Hiring	Communication				
nug22-sbb	0.0041	3.52	0.21	1	825.00	**0.0041**	3.52	0.21	1	62.52	1	0
nug24-sbb	*0.0030	9.73	1.06	1	86 445.00	**0.0030**	9.67	1.06	1	255.97	1	0
nug25-sbb	0.0039	17.10	0.15	1	21 509.00	**0.0039**	17.10	0.15	1	532.68	1	0
nug28-sbb	*0.0085	38.75	6.46	1	86 493.00	**0.0078**	40.80	0.60	1	592.98	1	0
nug30-sbb	*0.0150	81.08	11.86	2	86 469.00	**0.0150**	81.83	10.86	2	466.43	1	0
cms-1000	0.0962	169.56	0.00	3	10 594.00	0.1316	232.02	0.00	4	120.92	1	0
cms-1250	0.1782	314.02	0.00	5	80 472.00	0.2078	366.34	0.00	6	100.16	1	0
cms-1500	*0.4519	796.58	0.00	11	86 438.00	**0.3876**	683.20	0.00	10	14.61	1	0
modgen-real	0.0028	1.98	0.01	3	44.00	**0.0028**	1.98	0.01	2	6.03	1	0
raxml-real	0.0028	1.98	0.00	4	44.00	**0.0028**	1.98	0.00	4	7.07	1	0
nug22-sbb	0.0833	27.79	6.38	1	416.00	**0.0833**	7.51	0.84	1	67.26	0	1
nug24-sbb	0.0208	37.17	18.91	1	3637.00	**0.0208**	16.83	6.65	1	122.69	0	1
nug25-sbb	0.0167	36.12	28.97	1	4920.00	0.0200	24.71	9.62	1	182.51	0	1
nug28-sbb	0.0139	49.14	2762.95	1	72 359.00	**0.0139**	42.58	4.88	1	224.87	0	1
nug30-sbb	0.0238	104.24	23.72	2	72 428.00	**0.0238**	102.20	11.72	2	239.63	0	1
cms-1000	0.1250	182.22	0.00	3	27 558.00	0.1833	256.61	0.00	4	56.97	0	1
cms-1250	0.2083	343.42	0.00	5	20 081.00	0.2417	371.83	0.00	6	52.32	0	1
cms-1500	0.4167	683.20	0.00	10	31 815.00	**0.4167**	683.20	0.00	10	14.54	0	1
modgen-real	0.0417	2.88	0.05	1	61.00	**0.0417**	2.49	0.09	1	3.24	0	1
raxml-real	0.0417	5.76	45.70	1	61.00	**0.0417**	2.54	0.05	1	3.26	0	1
nug22-sbb	0.0437	3.52	0.21	1	621.00	**0.0437**	3.52	0.21	1	74.60	0.5	0.5
nug24-sbb	0.0119	9.67	1.06	1	909.00	**0.0119**	9.67	1.06	1	308.15	0.5	0.5
nug25-sbb	0.0103	17.10	0.15	1	818.00	**0.0103**	17.10	0.15	1	709.39	0.5	0.5
nug28-sbb	0.0108	40.80	0.60	1	12 525.00	**0.0108**	40.80	0.60	1	988.12	0.5	0.5
nug30-sbb	*0.0193	81.08	11.06	2	86 523.00	0.0195	84.06	10.25	2	857.82	0.5	0.5
cms-1000	0.1106	169.56	0.00	3	56 879.00	0.1413	211.65	0.00	4	200.12	0.5	0.5
cms-1250	*0.1975	329.00	0.00	5	86 422.00	0.2281	363.34	0.00	6	108.70	0.5	0.5
cms-1500	0.4021	683.20	0.00	10	64 928.00	**0.4021**	683.20	0.00	10	14.69	0.5	0.5
modgen-real	0.0222	1.98	0.01	1	72.00	**0.0222**	1.98	0.01	1	7.37	0.5	0.5
raxml-real	0.0222	1.98	0.01	1	80.00	**0.0222**	1.98	0.01	1	7.66	0.5	0.5
Average	0.0778	121.81	83.42	2.66	28 720.63	0.0823	125.20	1.70	3.01	186.90		

test is to analyze the performance of SciPhylomics using GraspCC-fed to dimension the virtual cluster in federated environments. Tests used the same application for generating phylogenomic trees of experiments of Section 6, Model Generator and RAxML, and two commercial clouds, Amazon EC2 and Microsoft Azure in the SciPhylomics execution.

Microsoft Azure also offer different types of virtual machines to be allocated by users for a specific workflow execution. We use the same virtual machines of Amazon EC2 described in Section 6 and the virtual machines of Microsoft Azure presented in Table 7 to perform these new experiments. The columns of the table represent the virtual machine name, the requirements of RAM memory, disk storage, gflop number and the financial cost to allocate this virtual machine per hour, respectively. Moreover, the communication costs of the providers were also considered now. The communication cost between the Amazon EC2 and Microsoft Azure machines is \$0.12 per GB of data transferred, whereas the costs per GB of data transmitted within the Amazon EC2 and the Microsoft Azure are \$0.1 and zero (0.0), respectively.

It is important to highlight that SciCumulus engine is not ready for a federated cloud execution. We performed an adaptation of SciCumulus engine by creating a shared storage that is accessed by all virtual machines in Amazon EC2 and Microsoft Azure environments. This shared storage was created using S3fs mechanism, as aforementioned. Although this adaptation allowed us to perform initial experiments, we are aware that it is an initial prototype and several modifications are still needed as we can state by analyzing results as presented in Fig. 11.

Fig. 11 shows the SciPhylomics executions using the GraspCC-fed solutions of the instances *modgen-real* and *raxml-real* of Table 6 with $\alpha_1 = 0.5$ and $\alpha_2 = 0.5$. These values were used because we wanted to consider equally the time and financial costs. These executions suggested that 1 VM of m3.xlarge type of Amazon, 1 VM of m3.2.xlarge type of Amazon and 1 VM of extraLarge type of Azure should be allocated per 1 h for each application, Model Generator and RAxML. The graphics present the amount of CAs (axis Y) executing in a given period of time (X axis). By analyzing the results presented in Fig. 11 we can state that GraspCC-fed indicated that we needed 3 virtual machines (47 virtual cores) for 1 h with financial cost of \$1.99 for each application, totaling 2 h and a financial cost of \$3.98, but the real execution using 3 virtual machines lasted for around 4 h (100% more) with an approximated financial cost of \$7.96. This behavior was due to problems identified in the adapted version of workflow engine (SciCumulus-fed). Although we have 47 virtual cores available for execution, SciCumulus did not use the entire set of available virtual machines during the entire execution course of the workflow. This happened because since we are using S3fs we have to face a severe delay when the data files are being synchronized. This way, in the beginning of the workflow execution, several CAs could not execute because the input data files were not available to

Table 7
Description of the Microsoft Azure virtual machines.

Virtual machine	RAM (GB)	Disk storage (GB)	GFLOP	Financial cost ($)
Small	1.75	197	23.46	0.06
Medium	3.5	262	46.93	0.12
Large	7	412	93.86	0.24
ExtraLarge	14	732	187.71	0.48

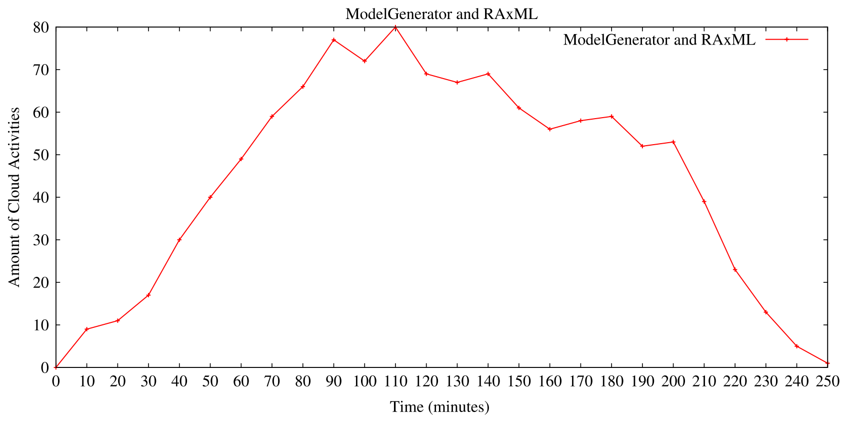


Fig. 11. Execution of ModelGenerator and RAxML in the Amazon EC2 and the Microsoft Azure using virtual machines given by the GraspCC-fed.

be consumed. SciCumulus-fed then waits until each data file is available in all virtual machines before starting a specific CA.

This way, the amount of CAs that are executing until 80 min is severely reduced and many cores remain idle. Another problem that occurred was related to the scheduling mechanism of SciCumulus. SciCumulus uses a cost model to distribute CAs in the several virtual machines. This cost model takes into account performance issues, financial cost and reliability but only for a single-provider cloud. Since we are executing in different clouds, communications issues impact more than execution time of the CA for several CAs. This way, SciCumulus scheduled CAs without taking into account the locality of these virtual machines, which requested more messages to be exchanged and impacted on the overall performance.

These results indicate that although the amount of virtual machines to allocate was suitable for this kind of experiment (*i.e.* GraspCC-fed produced a good estimative), modifications in the structure of the engine are still needed. In fact, we believe that in a near future all SWfMS have to be modified to consider federated cloud workflow executions. It seems to be inevitable that in the next few years several large-scale workflow execution will demand more computing power than a single cloud provider can offer, thus requiring federated cloud environments to execute in a feasible time.

9. Conclusion

Many large-scale scientific experiments use compute-intensive applications that consume and produce large datasets. These experiments are typically modeled as scientific workflows and they are executed in parallel on HPC environments such as clouds and more recently in federated cloud environments. Many SWfMS already support the execution of workflows in single-provider clouds and a few on federated clouds, and they focus on optimizing scheduling and data placement policies. However, most of them (with exception of SciCumulus) do not concern about optimizing the configuration of the virtual cluster in a single-cloud environment and none of them considers the virtual cluster dimensioning in federated clouds. In this article, we consider that the virtual cluster dimensioning is one important step in cloud-based workflow executions since it can negatively affect the performance and financial costs if the virtual cluster is under or over dimensioned.

This article introduces GraspCC-fed, an approach used for dimensioning the amount of virtual machines to allocate for a parallel workflow execution in single-provider cloud and federated cloud environments. GraspCC-fed is a Greedy Randomized Adaptive Search Procedure (GRASP) that estimates costs based on two factors: total workflow execution time and financial cost of the workflow execution. GraspCC-fed is responsible for determining the best configuration for the environment before a parallel workflow execution by using provenance information for the estimations. The result is the amount of virtual machines for each type, for each provider that should be allocated for the workflow execution subject to constraints set by the users such as maximum allowed execution time and maximum allowed financial costs.

First, in this article we compare a GRASP-based approach with a genetic algorithm-based approach for dimensioning the virtual cluster of workflows that execute in single-provider clouds. This first experiment aims at selecting the best approach to dimension federated cloud environments, which are environments that are clearly more complex. Since the GRASP-based approach outperformed the genetic algorithm-based approach, we extended the GRASP-based approach to consider parallel workflow executions in federated clouds. GraspCC-fed was then coupled to an adapted version of SciCumulus workflow engine for federated clouds, named SciCumulus-fed.

The benefits from investing on an optimal initial virtual cluster were evaluated in a series of experiments using a computing-intensive fragment of a real bioinformatics scientific workflow named SciPhylomics. GraspCC was able to reduce in 36% the total execution time of SciPhylomics in a single-provider cloud environment when compared to the genetic algorithm results. GraspCC-fed needed only around 7 s to generate the initial dimensioning of the virtual cluster before the workflow execution, which may be considered negligible when compared to the total execution time of the workflow.

When we used GraspCC-fed to dimension the virtual cluster in a federated cloud scenario, our estimate was influenced by several problems in the workflow engine, since the adaptations performed in SciCumulus have still to be improved. The data synchronization among all virtual machines allocated in different providers was higher than expected since the transfer rates informed by cloud providers were not 100% correct. In addition, the scheduling of SciCumulus-fed does not consider more than one cloud environment. This way, cloud activities were scheduled without taking into account in which provider the virtual machine was allocated.

Although we faced these several problems, the workflow execution in a federated cloud environment was acceptable (in terms of financial cost and total execution time) and this result reinforces the importance of dimensioning the virtual cluster to avoid unnecessary costs, especially in federated clouds where the number of variables involved in the dimensioning task grows exponentially.

Acknowledgments

The authors thank the Brazilian funding agencies CNPq, CAPES and FAPERJ. Daniel de Oliveira also thanks Microsoft for the Azure Educator Pass that allowed the authors to execute the experiments presented in this article.

References

[1] M. Mattoso, C. Werner, G.H. Travassos, V. Braganholo, E. Ogasawara, D.D. Oliveira, S.M. Cruz, W. Martinho, L. Murta, Towards supporting the life cycle of large scale scientific experiments, Int. J. Bus. Process Integr. Manag. 5 (1) (2010) 79–92.

[2] T. Hey, S. Tansley, K. Tolle, The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, Redmond, Washington, 2009, Eds..

[3] I.J. Taylor, E. Deelman, D.B. Gannon, Workflows for e-Science: Scientific Workflows for Grids, Springer, 2007.

[4] L. Grandinetti (Ed.), High Performance Computing and Grids in Action, in: Advances in Parallel Computing, 2008.

[5] J.M. Wozniak, T.G. Armstrong, K. Maheshwari, E.L. Lusk, D.S. Katz, M. Wilde, I.T. Foster, Turbine: a distributed memory dataflow engine for high performance many-task applications, Fund. Inform. J. 128 (3) (2013) 337–366.

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A.C. Laity, J.C. Jacob, D.S. Katz, Pegasus: a framework for mapping complex scientific workflows onto distributed systems, Sci. Program. 13 (3) (2005) 219–237.

[7] E.S. Ogasawara, J. Dias, V. Silva, F.S. Chirigati, D. de Oliveira, F. Porto, P. Valduriez, M. Mattoso, Chiron: a parallel engine for algebraic scientific workflows, Concurr. Comput.: Pract. Exper. 25 (16) (2013) 2327–2341.

[8] D. de Oliveira, E. Ogasawara, F. Baião, M. Mattoso, Scicumulus: a lightweight cloud middleware to explore many task computing paradigm in scientific workflows, in: 3rd International Conference on Cloud Computing, 2010, pp. 378–385.

[9] D. de Oliveira, K. Ocaña, E. Ogasawara, J. Dias, J. Gonçalves, F. Baião, M. Mattoso, Performance evaluation of parallel strategies in public clouds: a study with phylogenomic workflows, Future Gener. Comput. Syst. 29 (7) (2013) 1816–1825.

[10] M. Abouelhoda, S. Issa, M. Ghanem, Tavaxy: integrating taverna and galaxy workflows with cloud computing support, BMC Bioinformatics 13 (1) (2012) 77.

[11] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, I. Wang, Programming scientific and distributed workflow with Triana services, Concurr. Comput.: Pract. Exper. 18 (10) (2006) 1021–1037.

[12] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[13] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Grid Computing Environments Workshop, 2008, GCE'08, IEEE, 2008, pp. 1–10.

[14] L.M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds: towards a cloud definition, SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 50–55.

[15] R. Buyya, R. Ranjan, R. Calheiros, InterCloud: utility-oriented federation of cloud computing environments for scaling of application services, in: C.-H. Hsu, L. Yang, J. Park, S.-S. Yeo (Eds.), Algorithms and Architectures for Parallel Processing, in: Lecture Notes in Computer Science, vol. 6081, Springer, Berlin, Heidelberg, 2010, pp. 13–31.

[16] A. Celesti, F. Tusa, M. Villari, A. Puliafito, How to enhance cloud architectures to enable cross-federation, in: Proceedings of the 3rd International Conference on Cloud Computing, CLOUD'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 337–345.

[17] Amazon EC2, Amazon elastic compute cloud (Amazon EC2), 2010. http://aws.amazon.com/ec2/.

[18] Google Cloud Platform, December 05, 2013. https://cloud.google.com/products/compute-engine.

[19] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, M. Humphrey, Early observations on the performance of Windows Azure, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC'10, ACM, New York, NY, USA, 2010, pp. 367–376.

[20] B. Berriman, E. Deelman, G. Juve, M. Rynge, J.S. Vöckler, High-performance compute infrastructure in astronomy: 2020 is only months away, in: Proceedings of XXI Conference Astronomical Data Analysis Software and Systems, Vol. 461, Astronomical Society of the Pacific, 2012, pp. 91–94.

[21] F. Jrad, J. Tao, A. Streit, A Broker-based framework for multi-cloud workflows, in: Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds, MultiCloud'13, ACM, New York, NY, USA, 2013, pp. 61–68.

[22] J. Freire, D. Koop, E. Santos, C.T. Silva, Provenance for computational tasks: a survey, Comput. Sci. Eng. 10 (3) (2008) 11–21.

[23] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers, Examining the challenges of scientific workflows, Computer 40 (12) (2007) 24–32.

[24] D. de Oliveira, V. Viana, E. Ogasawara, K. Ocaña, M. Mattoso, Dimensioning the virtual cluster for parallel scientific workflows in clouds, in: Proceedings of the 4th ACM Workshop on Scientific Cloud Computing, Science, Cloud'13, ACM, New York, NY, USA, 2013, pp. 5–12.

[25] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, MIT Press, Cambridge, MA, USA, 1992.

[26] T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press, Oxford, UK, 1996.

[27] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, first ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[28] R. Coutinho, L. Drummond, Y. Frota, Optimization of a cloud resource management problem from a consumer perspective, in: Euro-Par 2013: Parallel Processing Workshops, in: Lecture Notes in Computer Science, vol. 8374, Springer, Berlin, Heidelberg, 2014, pp. 218–227.

[29] E. Ogasawara, D. De Oliveira, P. Valduriez, D. Dias, F. Porto, M. Mattoso, An algebraic approach for data-centric scientific workflows, Proc. VLDB 4 (11) (2011) 1328–1339.

[30] K. Ocaña, D. de Oliveira, E.S. Ogasawara, A.M.R. Dávila, A.A.B. Lima, M. Mattoso, SciPhy: a cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes, in: O.N. de Souza, G.P. Telles, M.J. Palakal (Eds.), BSB, in: Lecture Notes in Computer Science, vol. 6832, Springer, 2011, pp. 66–70.

[31] P.T. Endo, A.V. de Almeida Palhares, N.N. Pereira, G.E. Goncalves, D. Sadok, J. Kelner, B. Melander, J. Mangs, Resource allocation for distributed cloud: concepts and research challenges, IEEE Netw. 25 (4) (2011) 42–46.

[32] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: Proceedings of INFOCOM, IEEE, 2010, pp. 1–9.

[33] S. Pandey, L. Wu, S.M. Guru, R. Buyya, A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments, in: 24th International Conference on Advanced Information Networking and Applications (AINA), IEEE, 2010, pp. 400–407.

[34] Q. Li, Y. Guo, Optimization of resource scheduling in cloud computing, in: 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE, 2010, pp. 315–320.

[35] W.W. Adams, P. Loustaunau (Eds.), An Introduction to Gröbner Bases, Vol. 3, American Mathematical Society, 1994.

[36] J. Zhao, W. Zeng, M. Liu, G. Li, Multi-objective optimization model of virtual resources scheduling under cloud computing and it's solution, in: International Conference on Cloud and Service Computing (CSC), IEEE, 2011, pp. 185–190.

[37] M. Alicherry, T. Lakshman, Network aware resource allocation in distributed clouds, in: Proceedings of INFOCOM, IEEE, 2012, pp. 963–971.

[38] S. Chaisiri, B.-S. Lee, D. Niyato, Optimization of resource provisioning cost in cloud computing, IEEE Trans. Serv. Comput. 5 (2) (2012) 164–177.

[39] J. Baliga, R.W. Ayre, K. Hinton, R.S. Tucker, Green cloud computing: balancing energy in processing, storage, and transport, Proc. IEEE 99 (1) (2011) 149–167.

[40] F.A. de Oliveira Jr., T. Ledoux, Self-management of applications QoS for energy optimization in datacenters, in: Green Computing Middleware on Proceedings of the 2nd International Workshop, GCM 2011, ACM, New York, NY, USA, 2011, pp. 3:1–3:6.

[41] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.

[42] H. Goudarzi, M. Ghasemazar, M. Pedram, SLA-based optimization of power and migration cost in cloud computing, in: 12th International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2012, pp. 172–179.

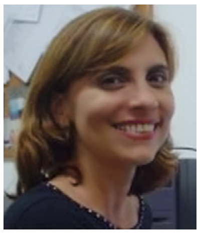
[43] H. Goudarzi, M. Pedram, Energy-efficient virtual machine replication and placement in a cloud computing system, in: 5th International Conference on Cloud Computing, IEEE, 2012, pp. 750–757.

[44] L. Xu, Z. Zeng, X. Ye, Multi-objective optimization based virtual resource allocation strategy for cloud computing, in: Proceedings of the 11th International Conference on Computer and Information Science, ICIS'12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 56–61.

[45] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC'12, IEEE Computer Society Press, Los Alamitos, CA, USA, 2012, pp. 22:1–22:11.

[46] Y. Zhao, Y. Zhang, W. Tian, R. Xue, C. Lin, Designing and deploying a scientific computing cloud platform, in: International Workshop on Grid Computing, 2012, pp. 104–113.
[47] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, M. Wilde, Swift: fast, reliable, loosely coupled parallel computation, in: IEEE Congress on Services, 2007, pp. 199–206.
[48] OpenNebula, 2013. http://www.opennebula.org.
[49] Q. Zhu, J. Zhu, G. Agrawal, Power-aware consolidation of scientific workflows in virtualized environments, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 1–12.
[50] J.A. Nelder, R. Mead, A simplex method for function minimization, Comput. J. 7 (4) (1965) 308–313.
[51] K. Deng, J. Song, K. Ren, A. Iosup, Exploring portfolio scheduling forlong-term execution of scientific workloads in IaaS clouds, in: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, ACM, New York, NY, USA, 2013, pp. 55:1–55:12.
[52] P. Lama, X. Zhou, AROMA: automated resource allocation and configuration of MapReduce environment in the cloud, in: Proceedings of the 9th International Conference on Autonomic Computing, ICAC'12, ACM, New York, NY, USA, 2012, pp. 63–72.
[53] P. Nguyen, M. Halem, A MapReduce workflow system for architecting scientific data intensive applications, in: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing, SECLOUD'11, ACM, New York, NY, USA, 2011, pp. 57–63.
[54] J. Wang, D. Crawl, I. Altintas, Kepler + Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems, in: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, WORKS'09, ACM, New York, NY, USA, 2009, pp. 12:1–12:8.
[55] D. Crawl, J. Wang, I. Altintas, Provenance for MapReduce-based data-intensive workflows, in: Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science, WORKS'11, ACM, New York, NY, USA, 2011, pp. 21–30.
[56] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J.M. Mellor-Crummey, B. Liu, S.L. Johnsson, Scheduling strategies for mapping application workflows onto the grid, in: HPDC, IEEE, 2005, pp. 125–134.
[57] K. Kennedy, M. Mazina, J.M. Mellor-Crummey, K.D. Cooper, L. Torczon, F. Berman, A.A. Chien, H. Dail, O. Sievert, D. Angulo, I.T. Foster, R.A. Aydt, D.A. Reed, D. Gannon, S.L. Johnsson, C. Kesselman, J. Dongarra, S.S. Vadhiyar, R. Wolski, Toward a framework for preparing and executing adaptive grid programs, in: IPDPS, IEEE Computer Society, 2002, p. 322.
[58] L. Bouganim, D. Florescu, P. Valduriez, Dynamic load balancing in hierarchical parallel database systems, in: Proceedings of the 22nd International Conference on Very Large Databases, 1996, pp. 436–447.
[59] D. de Oliveira, K. Ocaña, F.A. Baião, M. Mattoso, A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds, J. Grid Comput. 10 (3) (2012) 521–552.
[60] D. de Oliveira, K. Ocaña, E. Ogasawara, J. Dias, J. Gonćalves, M. Mattoso, Cloud-based phylogenomic inference of evolutionary relationships: a performance study, in: 3rd International Conference on Cloud Computing, 2012, pp. 378–385.
[61] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, H.T. Vo, VisTrails: visualization meets data management, in: SIGMOD International Conference on Management of Data, SIGMOD'06, ACM, New York, NY, USA, 2006, pp. 745–747.
[62] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, S. Mock, Kepler: an extensible system for design and execution of scientific workflows, in: Scientific and Statistical Database Management, 2004, pp. 423–424.
[63] M. Zvelebil, J. Baumb, Understanding Bioinformatics, first ed., Garland Science, 2007.
[64] A. Clark, Genomics of the evolutionary process 21 (6) (2006) 316–321.
[65] K. Katoh, H. Toh, Recent developments in the MAFFT multiple sequence alignment program, Brief. Bioinform. 9 (4) (2008) 286–298.
[66] K. Katoh, H. Toh, Parallelization of the MAFFT multiple sequence alignment program, Bioinformatics 26 (15) (2010) 1899–1900.
[67] T. Lassmann, E.L.L. Sonnhammer, Kalign—an accurate and fast multiple sequence alignment algorithm, BMC Bioinformatics 6 (2005) 298.
[68] J. Thompson, D. Higgins, T. Gibson, 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 (1994) 4673–4680.
[69] R.C. Edgar, Muscle: multiple sequence alignment with high accuracy and high throughput, Nucleic Acids Res. 32 (2004) 1792–1797.
[70] C.B. Do, M.S.P. Mahabhashyam, M. Brudno, S. Batzoglou, PROBCONS: probabilistic consistency-based multiple sequence alignment, Genome Res. 15 (2005) 330–340.
[71] T. Keane, C. Creevey, M. Pentony, T. Naughton, J. Mclnerney, 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 (1) (2006) 29.
[72] A. Stamatakis, RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models, Bioinformatics 22 (21) (2006) 2688–2690.
[73] B.E. Dutilh, V. van Noort, R.T.J.M. van der Heijden, T. Boekhout, B. Snel, M.A. Huynen, Assessment of phylogenomic and orthology approaches for phylogenetic inference, Bioinformatics 23 (7) (2007) 815–824.
[74] M. Resende, C. Ribeiro, Grasp with path-relinking: recent advances and applications, in: T. Ibaraki, K. Nonobe, M. Yagiura (Eds.), Metaheuristics: Progress as Real Problem Solvers, Springer, 2005, pp. 29–63.
[75] P. Festa, M.G.C. Resende, An annotated bibliography of GRASP—part I: algorithms, Int. Trans. Oper. Res. 16 (1) (2009) 1–24.
[76] P. Festa, M.G.C. Resende, An annotated bibliography of GRASP—part II: applications, Int. Trans. Oper. Res. 16 (2) (2009) 131–172.
[77] R. Coutinho, L. Drummond, Y. Frota, D. de Oliveira, K. Ocaña, Evaluating grasp-based cloud dimensioning for comparative genomics: a practical approach, in: Proceedings of the 2nd International Workshop on Parallelism in Bioinformatics, 2014, pp. 1–9.
[78] W. Banzhaf, F.D. Francone, R.E. Keller, P. Nordin, Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.
[79] K. Meffert, N. Rotstan, JGAP—Java genetic algorithms and genetic programming package, 2011, in: Available at: http://jgap.sf.net (accessed: 9 September 2014).
[80] V. Viana, D. de Oliveira, M. Mattoso, Towards a cost model for scheduling scientific workflows activities in cloud environments, in: World Congress on Services (SERVICES), IEEE Computer Society, 2011, pp. 216–219.
[81] S3fs, December 05, 2013. https://code.google.com/p/s3fs/wiki/FuseOverAmazon.
[82] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, Last 2010.
[83] A. Goncalves, L. Drummond, A. Pessoa, P. Hahn, Improving Lower Bounds for the Quadratic Assignment Problem by applying a Distributed Dual Ascent Algorithm, Technical Report, Cornell University Library, 2013.
[84] R. Burkard, S. Karisch, F. Rendi, QAPLIB – A Quadratic Assignment ProblemLibrary, J. Global Optim. 10 (4) (1997) 391–403.
[85] R. Adolphi, S. Spanier, The CMS experiment at the CERN LHC, CMS collaboration, J. Instrum. 3 (08) (2008) S08004.

Rafaelli de C. Coutinho is a Ph.D. student at the Institute of Computing at Fluminense Federal University (UFF). She graduated in Computer Science from UFF (2009) and obtained an M.S. also in Computer Science from the same university (2011). She has experience in Computer Science, and is interested in the following subjects: distributed algorithms, mathematical programming and cloud computing.


Lúcia M. A. Drummond graduated in Mathematics in Computing Modality from the State University of Rio de Janeiro in 1987, received the M.S. degree in Systems Engineering and Computer Science from the Federal University of Rio de Janeiro in 1990 and the Ph.D. degree also in Systems Engineering and Computer Science from the Federal University of Rio de Janeiro in 1994. She is currently an Associate Professor at the Institute of Computing of the Fluminense Federal University (UFF), engaged in the graduation and the graduate program, advising Master's and doctoral students.

Yuri Frota is a Professor of the Department of Computer Science at Fluminense Federal University, where he has been since 2010. He graduated in Computer Science from UECE (1999), obtained an M.S. in Computer Science from UFC (2002) and a Ph.D. also in Computer Science from UFRJ (2008). He has experience in computer science, focusing on algorithms, and engaged in the following subjects: combinatorial optimization and integer programming.

Daniel de Oliveira is a Professor of the Institute of Computing of the Fluminense Federal University (UFF) since 2013. He received the Doctor of Science degree from the Federal University of Rio de Janeiro (UFRJ) in 2012. His current research interests include scientific workflows, provenance, cloud computing, high performance computing and distributed and parallel databases. He participates in research projects in those areas, with funding from several Brazilian government agencies, including CNPq, CAPES and FAPERJ. He participates in several program committees of national and international conferences and workshops, and is a member of IEEE, ACM and of the Brazilian Computer Society.