# Multi-Objective Scheduling of Scientific Workflows in Multisite Clouds

Ji Liu, Esther Pacitti, Patrick Valduriez, Daniel de Oliveira, Marta Mattoso

# Multi-Objective Scheduling of Scientific Workflows in Multisite Clouds

Ji Liu[a], Esther Pacitti[a], Patrick Valduriez[a], Daniel de Oliveira[b], Marta Mattoso[c]

[a] *Inria, Microsoft-Inria Joint Centre, LIRMM and University of Monpellier, France*
[b] *Institute of Computing, Fluminense Federal University, Niteroi, Brazil*
[c] *COPPE, Federal University of Rio de Janeiro, Brazil*

## Abstract

Clouds appear as appropriate infrastructures for executing Scientific Workflows (SWfs). A cloud is typically made of several sites (or data centers), each with its own resources and data. Thus, it becomes important to be able to execute some SWfs at more than one cloud site because of the geographical distribution of data or available resources among different cloud sites. Therefore, a major problem is how to execute a SWf in a multisite cloud, while reducing execution time and monetary costs. In this paper, we propose a general solution based on multi-objective scheduling in order to execute SWfs in a multisite cloud. The solution includes a multi-objective cost model including execution time and monetary costs, a Single Site Virtual Machine (VM) Provisioning approach (SSVP) and ActGreedy, a multisite scheduling approach. We present an experimental evaluation, based on the execution of the SciEvol SWf in Microsoft Azure cloud. The results reveal that our scheduling approach significantly outperforms two adapted baseline algorithms (which we propose by adapting two existing algorithms) and the scheduling time is reasonable compared with genetic and brute-force algorithms. The results also show that our cost model is accurate and that SSVP can generate better VM provisioning plans compared with an existing approach.

*Keywords:* Scientific workflow; scientific workflow management system; multi-objective scheduling; parallel execution; multisite cloud

## 1. Introduction

Large-scale *in silico* scientific experiments typically take advantage of Scientific Workflows (SWfs) to model data operations such as loading input data, data processing, data analysis, and aggregating output data. SWfs enable scientists to model the data processing of these experiments as a graph, in which vertices represent data processing activities and edges represent dependencies between them. A SWf is the assembly of scientific data processing activities with data dependencies among them [15]. An activity is a description of a piece of work that forms a logical step within a SWf representation [22]. Since SWf activities may process big data, we can exploit data parallelism whereby one activity corresponds to several executable tasks, each working in parallel on a different part of the input data. Thus, a task is the representation of an activity within a one-time execution of this activity, which processes a data partition or chunk [22].

A SWf Management System (SWfMS) is the tool to manage SWfs [22]. In order to execute SWfs efficiently, SWfMSs typically exploit High Performance Computing (HPC) resources in a cluster, grid or cloud environment. Because of virtually infinite resources, diverse scalable services, stable quality of service and flexible payment policies, clouds have become an interesting solution for SWf execution. In particular, the user of Virtual Machines (VMs) makes it easy to deal with elasticity and workloads that change rapidly. A cloud is typically made of several sites (or data centers), each with its own resources and data. Thus, in order to use more resources than available at a single site or to access data at different sites, SWfs could also be executed in a distributed manner at different sites. Nowadays, the computing resources or data of a cloud provider such as Amazon or Microsoft are distributed at different sites and should be used during the execution of SWfs. As a result, a multisite cloud is an appealing solution for large scale SWf execution. As defined in [21], a multisite cloud is a cloud with multiple data centers, each at a different location (possibly in a different region) and being explicitly accessible to cloud users, typically in the data center close to them for performance reasons.

To enable SWf execution in a multisite cloud, the execution of each activity should be scheduled to a corresponding cloud site (or site for short). Then, the scheduling problem is to decide where to execute the activities. In general, to map the execution of activities to distributed computing resources is an NP-hard problem [36]. The objectives can be to reduce execution time or monetary cost, to maximize performance, reliability *etc.* Since the SWf execution may take a long time and cost much money, the scheduling problem may have multiple objectives, *i.e.* multi-objective. Thus, the multisite scheduling problem must take into account the impact of resources distributed at different sites, *e.g.* different bandwidths and data distribution at different sites, and different prices for VMs.

In this paper, we propose a general solution based on multi-objective scheduling in order to execute SWfs in a multisite cloud. The solution includes a multi-objective cost model, a Single Site VM Provisioning approach (SSVP) and ActGreedy, a multisite scheduling approach. The cost model includes two objectives, namely reducing execution time and monetary costs, under stored data constraints, which specify that some data should not be moved, because it is either too big or for proprietary reasons. Although useful for fixing some activities, these constraints do not reduce much the complexity of activity scheduling. We consider a homogeneous cloud environment, *i.e.* from single provider. The case of federated clouds (with multiple cloud providers) is beyond the scope of this paper and not a reality (although there are some recent proposals). ActGreedy handles multiple objectives, namely reducing execution time and monetary costs. In order to schedule a SWf in a multisite cloud, the SWf should be partitioned to SWf fragments, which can be executed at a single site. A SWf fragment (or fragment for short) is a subset of activities, dependencies and associated input data of the original workflow [21]. Then, each fragment can be scheduled by ActGreedy to the site that yields the minimum cost among all available sites. When a fragment is scheduled to a site, the execution of its associated activities is scheduled to the site. ActGreedy is based on our dynamic VM provisioning algorithm, called Single Site VM Provisioning (SSVP), which generates VM provisioning plans for the execution of fragments with minimum cost at the scheduled site based on a cost model. The cost model is used to estimate the cost of the execution of SWfs [14] according to a scheduling plan, which defines the schedule of fragments to execution sites. A VM provisioning plan defines how to provision VMs. For instance, it determines the types, corresponding number and the order of VMs to provision, for the execution of a fragment. The VM type determines some parameters such as the number of virtual CPUs, the size of memory and the default storage size of hard disk. The main contributions of this paper are:

1. The design of a multi-objective cost model that includes execution time and monetary costs, to estimate the cost of executing SWfs in a multisite cloud.
2. A single site VM provisioning approach (SSVP), to generate VM provisioning plans to execute fragments at each single site.
3. ActGreedy multisite scheduling algorithm that uses the cost model and SSVP to schedule and execute SWfs in a multisite cloud.
4. An extensive experimental evaluation, based on the implementation of our approach in Microsoft Azure, and using a real SWf use case (SciEvol [25], a bioinformatics Scientific workflow for molecular evolution reconstruction) that shows the advantages of our approach, compared with baseline algorithms.

This paper is organized as follows. Section 2 discusses related work. Section 3 introduces the problems for workflow scheduling. Section 4 describes the system architecture for SWf execution in a multisite cloud. Section 5 describes our multi-objective optimization approach. Section 6 describes our scheduling approaches including the SciEvol SWf use case, the approaches for SWf partitioning and three scheduling approaches, namely ActGreedy, LocBased and SGreedy. Section 7 is our experimental evaluation in Microsoft Azure cloud [2]. Section 8 concludes.

## 2. Related Work

To the best of authors' knowledge, there is no solution to execute SWfs in a multisite cloud environment that takes into account both multiple objectives and dynamic VM provisioning. The related work either focuses on static VM provisioning [16], single objective [9, 23, 29, 32, 34, 37, 35, 17, 17, 23, 21] or single site execution [14, 18, 30]. Static VM provisioning refers to the use of the existing VMs (before execution) for SWf execution without changing the types of VMs during execution. However, existing cost models are not suitable for the SWfs that have a big part of the sequential workload. For instance, the dynamic approach proposed in [13] ignores the sequential part of the SWf and the cost of provisioning VMs, which may generate VM provisioning plans that yield high cost.

Many solutions for workflow scheduling [9, 23, 29, 32, 34, 37] focus on a single objective, *i.e.* reducing execution time. These solutions address the scheduling problem in a single site cloud. Classic heuristics have been used in scheduling algorithms, such as HEFT [35], min-min [17], max-min [17] and Opportunistic Load Balancing (OLB) [23], but they only address the single objective. Furthermore, they are designed for static computing resources in grid or cluster environments. In contrast, our algorithm handles multiple objectives, which are reducing execution time and monetary costs, with dynamic VM provisioning support. Although some general heuristics, *e.g.* genetic algorithms [35], can generate near optimal scheduling plans, it is not always feasible to design algorithms for every possible optimization problem [35] and it is not trivial to configure parameters for the problem. In addition, its may take much time to generate scheduling plans. A brute-force method can generate an optimal scheduling plan, but its complexity is very high.

Some multi-objective scheduling techniques [14, 18, 30] have been proposed. However, they do not take the distribution of resources at different sites into consideration, so they are not suitable for a multisite environment. De Oliveira *et al.* [14] propose a greedy scheduling approach for the execution of SWfs at a single site. However, this approach is not appropriate for multisite execution of SWfs as it schedules the most suitable activities to each VM, which may incur transferring of big data transfer. Rodriguez and Buyya [30] introduce an algorithm for scheduling dynamic bags of tasks and dynamic VM provisioning for the execution of SWfs with multiple objectives in a single site cloud. Rather than using real execution, they simulate the execution of SWfs , thus missing the heterogeneity among the activities of the same SWf, to evaluate their proposed approaches. In real SWf execution, the activities generally correspond to different programs to process data. However, in simulations of SWf execution, the activities are typically made homogeneous, namely, they correspond to the same program. Different from the existing approaches, our approach is suitable for multisite execution and is evaluated by executing a real-life SWf on a multisite cloud that is Azure.

Some scheduling techniques have been proposed for the multisite cloud, yet focusing on a single objective, *i.e.* reducing execution time. For instance, Liu *et al.* [21] present a workflow partitioning approach and data location based scheduling approach. But this approach does not take monetary cost into consideration. Our approach uses an *a priori* method, where preference information is given by users and then the best solution is produced. Our approach is based on a multi-objective scheduling algorithm focusing on minimizing a weighted sum of objectives. The advantage of such approach is that it is automatically guided by predetermined weights while the disadvantage is that it is hard to determine the right values for the weights [10]. In contrast, *a posteriori* methods produce a Pareto front of solutions without predetermined values [10]. Each solution is better than the others with respect to at least one objective and users can choose one from the produced solutions. However, this method requires users to pick the most suitable solution. In this paper, we assume that users have a clear idea of the importance of objectives, and they can determine the value for the weight of each objective. One advantage of using *a priori* method is that we can produce optimal or near optimal solutions without user interference at run-time. When we are using the method of Pareto front, several solutions may be produced to be chosen by the user. Finally, when the weight of each objective is positive, the minimum of the sum is already a Pareto optimal solution [38] [24] and our proposed approach can generate a Pareto optimal or near-optimal solution with the predefined weights. Therefore, we do not consider *a posteriori* methods.

The existing cost models for generating VM provisioning plans [13] are not suitable for SWfs that have some sequential part in their workload. For instance, as presented in [13], the real execution time of two SWfs (SciEvol and SciPhylomics [27]) in Amazon EC2 [1] is two times the estimated time. In addition, some cost models [14][31] for SWf scheduling cannot be used for generating a proper number of virtual CPUs (CPUs designed to VMs) to instantiate for SWf execution at a multisite cloud. The instantiation of virtual CPUs is realized by provisioning corresponding VMs. Our cost model does consider the cost to provision VMs and the sequential parts of the workload in SWf execution. Furthermore, it can be used to estimate the optimal number of virtual CPUs to instantiate, which is used to generate different provisioning plans for different weights of objectives.

Duan *et al.* [16] propose a multisite multi-objective scheduling approach with consideration of different bandwidths in a multisite environment. However, it is only suitable for static computing resources. Coutinho *et al.* [13] propose a GraspCC algorithm to generate a provisioning plan for fragment execution. However, GraspCC relies on the strong assumption that the entire workload of SWfs can be executed in parallel. Furthermore, it cannot reuse existing started VMs and its cost model is too simple, *e.g.* does not consider the cost of starting VMs, which may be high with many VMs to provision. In our VM provisioning approach, we use a more precise cost model, assuming that part of the workload can be executed only sequentially. We also consider the existing started VMs and the cost to start VMs before fragment execution.

## 3. Problem Definition

This section introduces some important terms, *i.e.* SWf, SWf fragment and multisite cloud and defines the scheduling problem in the multisite cloud.

A *SWf* is described as a Directed Acyclic Graph (DAG) denoted by $W(V,E)$. Let $V = \{v_1, v_2, ..., v_n\}$ be a set of vertices, which are associated with the scientific data processing activities and $E = \{e_{i,j}$: $v_i, v_j \in V$ and $v_j$ consumes the output data of $v_i$ $\}$ be a set of edges that correspond to dependencies between activities in $V$. Activity $v_j$ is the following activity of Activity $v_i$ and Activity $v_i$ is a preceding activity of Activity $v_j$. The dependencies can be data or control dependencies. Compared to data dependencies, fewer data are transferred in control dependencies. The transferred data in a control dependency is the configuration parameters for activity execution while the transferred data in a data dependency is the input data to be processed by the following activity. The activity that processes control parameters is a control activity. Since the control activity takes little time to execute, we assume that a control activity has no workload. In addition, we assume that the data stored at a specific site may not be allowed to be transferred to other sites because of proprietary or big amounts of data, which is denoted as *stored data constraint*. If an activity needs to read the data from the stored data located at a specific site, this activity is denoted as *fixed activity*.

A large-scale SWf and its input data can be partitioned into several fragments [12] [21]. Thus, a SWf can be described as the assembly of fragments and fragment dependencies, *i.e.* $W(WF, FE)$ where $WF = \{wf_1, wf_2, ..., wf_n\}$ represents a set of fragments connected by dependencies in the set $FE = \{fe_{i,j}$: $wf_i, wf_j \in WF$ and $wf_j$ consumes the output data of $wf_i\}$. A fragment dependency $fe_{i,j}$ represents that fragment $wf_j$ processes the output data of fragment $wf_i$. $fe_{i,j}$ is the input dependency of $wf_j$ and output dependency of $wf_i$. A fragment can be denoted by $wf(V, E, D)$. $V$ represents the activities, $E$ represents the dependencies and $D$ represents the input data of the workflow fragment.

We denote the SWf execution environment by a configured multisite cloud[1] $MS(S)$, which consists of a set of sites $S$. A multisite cloud configuration defines the instances of VMs and storage resources for cloud users in a multisite cloud. One site $s_i \in S$ is composed of a set of Web domains. A Web domain contains a set of VMs, shared storage resources, and stored data. In this paper, we assume that one site contains only one Web domain for the execution of a SWf. We assume that the available VMs for the execution of SWfs in a multisite cloud have the same virtual CPUs, *i.e.* the virtual CPUs have the same computing capacity, but the number of virtual CPUs in each VM may be different. In addition, we assume that the price to instantiate VMs of the same type are the same at the

[1] The multisite cloud environment configured for the quota of resources that can be used by a cloud user.

same site while the prices at different sites can be different. The price is the monetary cost to use a VM during a time quantum (the quantum varies according to the cloud provider, *e.g.* one hour or one minute). Time quantum is the smallest possible discrete unit to calculate the cost of using a VM. For instance, if the time quantum is one minute and the price of a VM is 0.5 dollar per hour, the cost to use the VM for the time period of $T$ ($T \geq N - 1$ minutes and $T < N$ minutes) will be $\frac{N*0.5}{60}$ dollars.

Scheduling fragments requires choosing a site to execute a fragment, *i.e.* mapping each fragment to an execution site. A fragment scheduling plan defines the map of fragments and sites. When a fragment is scheduled at a site, the activities of the fragment are also scheduled at that site. Based on a multi-objective cost model, the problem we address has the following form [28]:

$\min(Cost(Sch(SWf, S)))$
subject to
stored data constraint

The decision variable is $Schedule(wf, s)$, which is defined as

$$Schedule(wf, s) = \begin{cases} 1 \text{ if Fragment } wf \text{ is scheduled at Site } s \\ 0 \text{ otherwise} \end{cases}$$

Thus, the scheduling problem is, given a multi-objective cost model, how to generate a fragment scheduling plan $Sch(SWf, S)$, for which the corresponding SWf execution has minimum $Cost(Sch(SWf, S))$ while respecting the stored data constraints $Const(data)$ with $data \in input(SWf)$. The cost is the value calculated based on formulas defined in a cost model, *e.g.* Formulas 5.1.1 and 5.1.2, which depends on a scheduling plan and VM provisioning plans at scheduled sites. In the scheduling plan, for each Fragment $wf$ and site $s$, only if Fragment $wf$ is scheduled at Site $s$, the decision variable is 1; otherwise, the variable is 0. One fragment can be scheduled at only one site. The search space of scheduling plans contains all the possible scheduling plans, *i.e.* for any combination of $wf$ and $s$, we can find a scheduling plan in the search space that contains the decision variable $Schedule(wf, s) = 1$. If the cost is composed of just one objective, the problem is a single objective optimization problem. Otherwise, the problem is a multi-objective problem. The cost model is detailed in Section 5.1. In the paper, we use SSVP (see Section 5.2) to generate VM provisioning plans. The stored data constraints can be represented as a matrix (as the one presented below), and its cell values are known before execution, where each row $a_i$ represents an activity, each column $s_j$ represents a cloud site, and $(a_i, s_j) = 1$ means that $a_i$ needs to read the data stored at $s_j$.

$$\begin{array}{cccc} & s_1 & s_2 & s_3 \\ a_1 & 1 & 0 & 0 \\ a_2 & 0 & 0 & 1 \\ a_3 & 0 & 1 & 0 \end{array}$$

## 4. Multisite SWfMS Architecture

In this section, we present the architecture of a multisite SWfMS. This architecture (see Figure 1) has four modules: workflow partitioner, multisite scheduler, single site initialization, and single site execution. The workflow partitioner partitions a SWf into fragments (see Section 6.2). After SWf partitioning, the fragments are scheduled to sites by the multisite scheduler. After scheduling, in order to avoid restarting VMs for the execution of continuous activities, all the activities scheduled at the same site are grouped as a fragment to be executed. Then, the single site initialization module prepares the execution environment for the fragment, using two components, *i.e.* VM provisioning and multisite data transfer. At each site, the VM provisioning component deploys and initializes VMs for the execution of SWfs. The deployment of a VM is to create a VM under a user account in the cloud. The deployment of the VM defines the type and location, namely the cloud site, of the VM. The initialization of a VM is the process of starting the VM, installing programs and configuring parameters of the VM, so that the VM can be used for executing the tasks of fragments. The multisite data transfer module transfers the input data of fragments to the site. Finally, the single site execution module starts the execution of the fragments at each site. This can be realized by an existing single site SWfMS, *e.g.* Chiron [26]. Within a single site, when the execution of its fragment is finished and the output data is moved to other sites, the VMs are shut down. When the execution of the fragment is waiting for the output data produced by other sites and the output data at this site are transferred to other corresponding sites, the VMs are also shut down to avoid the useless monetary cost. When the necessary data is ready, the VMs are restarted to continue the execution of the fragment.

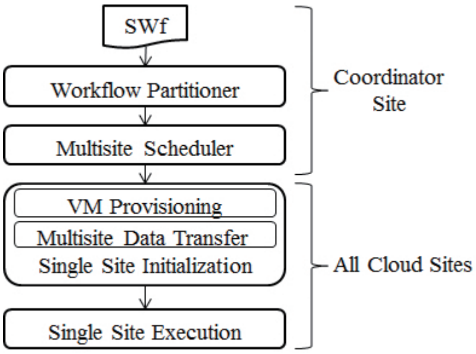


Figure 1: **System Architecture.**

In a multisite cloud, there are two types of sites, *i.e.* coordinator and participant. The coordinator is responsible for coordinating the execution of fragments at different participants. Two modules, namely workflow partitioner and multisite scheduler, are implemented at the coordinator site. Both the coordinator and participants execute the scheduled fragments. The initialization module and single site execution module are implemented at both the coordinator and participants.

## 5. Multi-objective Optimization

This section focuses on multi-objective optimization, which is composed of a multi-objective cost model, used to estimate the cost of executing SWfs in a multisite cloud, our algorithm (SSVP) to generate VM provisioning plans to execute fragments at each single site and a cost estimation method for the scheduling process.

### *5.1. Multi-objective Cost Model*

We propose a multi-objective cost model, which is composed of time cost, *i.e.* execution time, and monetary cost for the execution of SWfs. In order to choose a good scheduling plan, we need a cost model to estimate the cost of executing a SWf in a multisite cloud. A cost model is composed of a set of formulas to estimate the cost of the execution of SWfs [14] according to a scheduling plan. It is generally implemented in the scheduling module and under a specific execution environment. In the case of this paper, the execution environment is a multisite cloud. Our proposed cost model is an extension of the model proposed in [14] and [31]. In addition, the cost model is also used to calculate the real cost by replacing estimated parameters by real values obtained from the real execution in the evaluation part, *i.e.* Section 7.

The cost of executing a SWf can be defined by:

$$\begin{aligned} Cost(Sch(SWf,S)) =& \omega_t * \frac{Time(Sch(SWf,S))}{DesiredTime} \\ &+ \omega_m * \frac{Money(Sch(SWf,S))}{DesiredMoney} \end{aligned} \quad (5.1.1)$$

, where $DesiredTime$ represents the desired execution time to execute the SWf and $DesiredMoney$ is the desired monetary cost for the execution. Both $DesiredTime$ and $DesiredMoney$ are configured by the users. Note that these may be unfeasible to obtain for the execution of the SWf. We take the desired execution time and monetary costs into consideration in the cost model while the real execution time and monetary costs may be bigger or smaller depending on the real execution environment. $Time(SWf)$ and $Money(SWf)$ is the real execution time and real monetary cost for the execution of the SWf. $\omega_t$ and $\omega_m$ represent the weights for execution time and monetary costs, which are positive.

However, it is difficult to estimate the execution time and monetary costs for the whole SWf even with a scheduling plan according to Formula 5.1.1 since it is hard to generate a VM provisioning plan for each site with global desired execution time and monetary costs. As shown in Formula 5.1.2, we decompose the cost model as the sum of the cost of executing each fragment.

$$Cost(Sch(SWf,S)) = \sum_{wf_i \in SWf}^{Schedule(wf_i,s_j)=1} Cost(wf_i, s_j) \quad (5.1.2)$$

The cost of executing a fragment at a site can be defined as:

$$\boxed{Cost(wf,s) = \omega_t * Time_n(wf,s) + \omega_m * Money_n(wf,s)} \quad (5.1.3)$$

The box represents that the formula is referred in the following sections and the meaning of boxes of other formulas are the same. $\omega_t$ and $\omega_m$, which are the same as that in Formula 5.1.1, represent the weights for the execution time and the monetary cost to execute Fragment $wf$ at Site $s$. $Time_n(wf,s)$ and $Money_n(wf,s)$ are normalized values that are defined in Sections 5.1.1 and 5.1.2. Since the value of time and money is normalized, the cost has no unit. In the rest of this paper, cost represents the normalized cost, which has no real unit. And we use SSVP (see Section 5.2) to generate VM provisioning plans at each site for the execution of SWf fragments.

#### *5.1.1. Time Cost*

In this section, we present the method to estimate the time to execute Fragment $wf$ at Site $s$ with scheduling plan $SP$. The normalized time cost used in Formula 5.1.3 can be defined as:

$$\boxed{Time_n(wf,s) = \frac{Time(wf,s)}{DesiredTime(wf)}} \quad (5.1.4)$$

, where $Time(wf,s)$ represents the entire time for the execution of Fragment $wf$ at Site $s$ and $DesiredTime(wf)$ is the desired time to execute Fragment $wf$. Assuming that each activity has a user estimated workload $Workload(a, inputData)$ with a specific amount of input data $inputData$, we can calculate the desired execution time of Fragment $wf$ with the user defined desired time for the whole SWf by Formula 5.1.5.

$$\boxed{\begin{aligned} DesiredTime(wf) =& \frac{\sum_{a_i \in CP(wf)} workload(a_i, inputData)}{\sum_{a_j \in CP(SWf)} workload(a_j, inputData)} \\ &* DesiredTime \end{aligned}} \quad (5.1.5)$$

In this formula, $CP(SWf)$ represents the critical path of Workflow $SWf$, which can be generated by the method proposed by Chang *et al.* [11]. A critical path is a path composed of a set of activities with the longest average execution time from the start activity to the end activity [11]. In a SWf, the start activity is the activity that has no input dependency and the end activity is the activity that has no output dependency. Similarly, $CP(wf)$ represents the critical path of Fragment $wf$. The workload $workload(a_i, inputData)$ of an activity $a_i$ with a specific amount of data $inputData$ is estimated by users according to the features of the SWf. $DesiredTime$ is the desired execution time for the whole workflow, defined by user. Since the time to execute a fragment or a SWf is similar to that of the executing the activities in the critical path, we calculate the desired time for a fragment as the part of the time to execute the same workload of activities in the critical path of the SWf as that of the fragment.

In order to execute Fragment $wf$ at Site $s$, the system needs to initialize the corresponding execution site, to transfer the corresponding input data of Fragment $wf$ to Site $s$ and to run the program in the VMs of the site. The initialization of the site is explained in Section 4. This way, the entire time for the execution of Fragment $wf$ at Site $s$ can be estimated by the following formula:

$$\boxed{\begin{aligned} Time(wf,s) =& InitializationTime(wf,s) \\ &+ TransferTime(wf,s) \\ &+ ExecutionTime(wf,s) \end{aligned}} \quad (5.1.6)$$

, where $s$ represents the site to execute Fragment $wf$ according to the scheduling plan $SP$, $InitializationTime$ represents the time to initialize Site $s$, $TransferTime$ is the time to transfer input data from other sites to Site $s$ and $ExecutionTime$ is the time to execute the fragment. In order to simplify the problem, we ignore the cost (both time cost and monetary cost) to restart VMs at a site to wait for the output data produced by other sites. In this formula, the time to wait for the input data produced by the activities executed at another site (Site $s_o$) is not considered since this time is considered in that of the fragment executed at Site $s_o$.

The multisite SWfMS needs to provision $m$ (determined by SSVP) VMs to execute Fragment $wf$ at a single site. As explained in Section 4, to provision a VM is to deploy and to initialize a VM at a cloud site. The time to provision the VMs at a single site is estimated by Formula 5.1.7.

$$\boxed{InitializationTime(wf, s) = m * InitializationTime} \quad (5.1.7)$$

$InitializationTime$ represents the average time to provision a VM. The value of $InitializationTime$ can be configured by users according to the cloud environment, which can be obtained by measuring the average time to start, install the required programs and configure 2 - 3 VMs. We assume that there is only one VM being started at a Web domain at the same time, which is true in Azure.

The time for data transfer is the sum of the time to transfer input data stored in other sites to Site $s$. The data transfer time can be estimated by formula 5.1.8.

$$\boxed{TransferTime(wf, s) = \sum_{s_i \neq s} \frac{DataTransferAmount(wf, s_i)}{DataTransferRate(s_i, s)}} \quad (5.1.8)$$

$DataTransferAmount(wf, s_i)$ is the amount of input data of Fragment $wf$ stored at Site $s_i$, which is defined later (see Formula 5.1.9). $DataTransferRate(s_i, s)$ represents the data transfer rate between Site $s_i$ and Site $s$, which can be roughly obtained by measuring the amount of data transferred by Linux SCP command during a specific period of time between two VMs located at the two sites.

We assume that the amount of input data for each dependency is estimated by the user. The amount of data to be transferred from another site ($s_i$) to the site ($s$) to execute the fragment can be estimated by Formula 5.1.9.

$$\begin{aligned} &DataTransferAmount(wf, s_i) \\ &= \sum_{a_j \in wf} \sum_{a_k \in preceding(a_j)}^{a_k \in activities(s_i)} AmountOfData(e_{k,j}) \end{aligned} \quad (5.1.9)$$

where $preceding(a_j)$ represents the preceding activities of Activity $a_j$ at Site $s_o$. $s_i$ represents the site that originally stores a part of the input data of Fragment $wf$. $activities(s_i)$ represents the activities in the fragments that are scheduled at Site $s_i$. As defined in Section 3, $e_{k,j}$ represents the data dependency between Activity $a_k$ and $a_j$.

Assuming that one site has $n$ (determined by SSVP) virtual CPUs to execute a fragment, according to Amdahl's law [33], the execution time can be estimated by Formula 5.1.10.

$$\boxed{\begin{aligned} &ExecutionTime(n, wf, s) \\ &\quad = \frac{(\frac{\alpha}{n} + (1-\alpha)) * Workload(wf, InputData)}{ComputingSpeedPerCPU} \end{aligned}} \quad (5.1.10)$$

$\alpha$[1] represents the percentage of the workload that can be executed in parallel. $ComputingSpeedPerCPU$[2] represents the average computing performance of each virtual CPU, which is measured by FLOPS (FLoating-point Operations Per Second) [13]. $Workload$ represents the workload of Fragment $wf$ with specific amounts of input data $InputData$, which can be measured by the number of FLOP (FLoat-point Operations) [13]. $\alpha$, the function of $Workload$ and the parameter $ComputingSpeedPerCPU$ should be configured by the user according to the features of site and the SWf to be executed. In this paper, we calculate the workload of a fragment by the following function:

$$Workload(wf, InputData) = \sum_{a_j \in wf} workload(a_j, inputData) \quad (5.1.13)$$

The workload of an activity with a specific amount of input data is estimated according to the SWf.

The parameters $n$ and $m$ can be determined by a dynamic VM provisioning algorithm, which is detailed in Section 5.2.

### *5.1.2. Monetary Cost*

In this section, we present the method to estimate the monetary cost to execute Fragment $wf$ at Site $s$ with a scheduling plan $SP$. The normalized monetary cost used in Formula 5.1.3 can be defined by the following formula:

$$\boxed{Money_n(wf, s) = \frac{Money(wf, s)}{DesiredMoney(wf)}} \quad (5.1.14)$$

Let us assume that each activity has a user defined workload $Workload(a, inputData)$ similar to that of time cost. Inspired by Fard *et al.* [18], we calculate the desired monetary cost of executing a fragment $wf$ by Formula 5.1.15, which is the part

[1] $\alpha$ can be obtained by measuring the execution time of executing the fragment with a small amount of input data two times with different number of virtual CPUs. For instance, assume that we have $t_1$ for $n$ virtual CPUs and $t_2$ for $m$ virtual CPUs,

$$\alpha = \frac{m * n * (t_2 - t_1)}{m * n * (t_2 - t_1) + n * t_1 - m * t_2} \quad (5.1.11)$$

[2] According to [4], we use the following formula to calculate the computing speed of a virtual CPU. The unit of CPU Frequency is GHz and the unit of Computing speed is GFLOPS.

$$ComputingSpeedPerCPU = 4 * CPUFrequency \quad (5.1.12)$$

of the monetary cost to execute the workload of Fragment $wf$ in the SWf. In Formula 5.1.15, $a_i$ and $a_j$ represent an activity.

$$\boxed{\begin{aligned} DesiredMoney(wf) = & \frac{\sum_{a_i \in wf} workload(a_i, inputData)}{\sum_{a_j \in SWf} workload(a_j, inputData)} \\ & * DesiredMoney \end{aligned}} \tag{5.1.15}$$

Similar to Formula 5.1.6 for estimating the time cost, the monetary cost also contains three parts, *i.e.* site initialization, data transfer and fragment execution, as defined in Formula 5.1.16.

$$\boxed{\begin{aligned} Money(wf, s) = & InitializationMoney(wf, s) \\ & + TransferMoney(wf, s) \\ & + ExecutionMoney(wf, s) \end{aligned}} \tag{5.1.16}$$

where $s$ represents the site to execute Fragment $wf$. *InitializationMoney* represents the monetary cost to provision the VMs at Site $s$, $TransferMoney$ is the monetary cost to transfer input data of Fragment $wf$ from other sites to Site $s$ and *ExecutionMoney* is the monetary cost to execute the fragment.

The monetary cost to initialize a single site is estimated by Formula 5.1.17, *i.e.* the sum of the monetary cost for provisioning each VM.

$$\boxed{\begin{aligned} & InitializationMoney(wf, s) = \\ & \sum_{i=1}^{m} (MonetaryCost(VM_i, s) * \frac{(m - i) * InitializationTime}{TimeQuantum}) \end{aligned}} \tag{5.1.17}$$

$MonetaryCost(VM_i, s)$ is the monetary cost to use a VM $VM_i$ per time quantum at Site $s$. *InitializationTime* represents the average time to provision a VM. $TimeQuantum$ is the time quantum in the cloud. $m$ (determined by SSVP) represents that there are $m$ VMs to execute Fragment $wf$. Similar to the time cost estimation, we assume that there is only one VM being started at a Web domain at the same time. In addition, during the provisioning of VMs at a single site, the VM that has less virtual CPUs is provisioned first in order to reduce the monetary cost for waiting for the provisioning of other VMs. Thus, the order of $VM_i$ is also in this order in Formula 5.1.17, *i.e.* $VM_i$ begins with the VM that has less virtual CPUs.

The monetary cost for data transfer should be estimated based on the amount of transferred data and the price to transfer data among different sites, which is defined by the cloud provider. In this paper, the monetary cost of data transfer is estimated according to Formula 5.1.18, where *DataTransferUnitCost* represents the monetary cost to transfer a unit, *e.g.* gigabyte(GB), of data from the original site ($s_o$) to the destination site ($s$). *DataTransferUnitCost* is provided by the cloud provider.

$$\boxed{\begin{aligned} TransferMoney(wf, s) = & \sum_{s_i \neq s} (DataTransferAmount(wf, s_i) \\ & * DataTransferUnitCost(s_i, s)) \end{aligned}} \tag{5.1.18}$$

$DataTransferAmount(wf, s_i, s)$ is defined in Formula 5.1.9.

The monetary cost for the fragment execution can be estimated by Formula 5.1.19, *i.e.* the monetary cost of using $n$ virtual CPUs during the Fragment execution.

$$\boxed{\begin{aligned} & ExecutionMoney(n, wf, s) \\ & = n * MCostPerCPU(s) * \lfloor \frac{ExecutionTime(n, wf, s)}{TimeQuantum} \rfloor \end{aligned}} \tag{5.1.19}$$

$ExecutionTime(n, wf, s)$ is defined in Formula 5.1.10. The parameter *MCostPerCPU* represents the average monetary cost to use one virtual CPU in one time quantum at Site $s$, which can be the price of VMs divided by the number of virtual CPUs. We assume that the monetary cost of each virtual CPU in the VMs of available different types are the same at a site. $TimeQuantum$ represents the time quantum in the cloud.

The original parameters mentioned in this section are listed in Table 1. The other parameters that are not listed in Table 1 are derived based on the listed original parameters.

### 5.2. Single Site VM Provisioning

We propose a single site VM provisioning algorithm, called SSVP, to generate VM provisioning plans. In order to execute a fragment at a site, the multisite SWfMS system needs to provision a set of VMs to construct a cluster at a site. The problem of how to provision VMs, *i.e.* to determine the number, type and order of VMs to provision, is critical to the cost of workflow execution.

Based on the aforementioned formulas, we can calculate the execution cost to execute Fragment $wf$ at Site $s$ without considering the cost of site initialization and data transfer according to Formula 5.2.1. This formula is used to calculate an optimal number, which is used to generate a VM provisioning plan in SSVP, of virtual CPUs to instantiate for the execution of fragments.

$$\begin{aligned} ExecutionCost(N, wf, s) = & \omega_t * \frac{ExecutionTime(n, wf, s)}{DesiredTime(wf)} \\ & + \omega_m * \frac{ExecutionMoney(n, wf, s)}{DesiredMoney(wf)} \end{aligned} \tag{5.2.1}$$

In Formula 5.2.1, $ExecutionTime(n, wf, s)$ is defined in Formula 5.1.10, $DesiredTime(wf)$ is defined in Formula 5.1.5, $ExecutionMoney(n, wf, s)$ is defined in Formula 5.1.19 and $DesiredMoney(wf)$ is defined in Formula 5.1.15. In order to get a general formula to calculate the optimal number of virtual CPUs, we use Formula 5.2.2, which has no floor function, for $ExecutionMoney(n, wf, s)$.

$$\begin{aligned} & ExecutionMoney(n, wf, s) \\ & = n * MCostPerCPU(s) * \frac{ExecutionTime(n, wf, s)}{TimeQuantum} \end{aligned} \tag{5.2.2}$$

Finally, the execution cost can be expressed as Formula 5.2.3 with the parameters defined in Formula 5.2.4.

Table 1: **Parameter summary.** Original represents where the value of the parameter comes from. UD: that the parameter value is defined by users; ESWf: that the parameter value is estimated according to the SWf; Measure: that the parameter value is measured by user with the SWf and in a cloud environment; Cloud: the parameter value is obtained from the cloud provider; Execution: measured during the execution of SWf in a multisite cloud.

| Parameter | Meaning | Original |
|---|---|---|
| DesiredTime | Desired execution time | UD |
| DesiredMoney | Desired monetary cost | UD |
| workload | The workload of an activity | ESWf |
| AmountOfData | The amount of data in a data dependency | ESWf |
| InitializationTime | The time to initialize a VM | Measure |
| DataTransferRate | Data transfer rate between two sites | Measure |
| $\alpha$ | The percentage of the workload that can be executed in parallel | Measure |
| CPUFrequency | Computing performance of virtual CPUs | Cloud |
| MonetaryCost | Monetary cost of a VM | Cloud |
| TimeQuantum | The time quantum of a cloud | Cloud |
| DataTransferUnitCost | The monetary cost to transfer a unit of data between two sites | Cloud |
| MCostPerCPU | The monetary cost to use a virtual CPU at a site | Cloud |
| ExecutionTime | The execution time of a fragment at a site | Execution |

$$ExecutionCost(N, wf, s) = a * n + \frac{b}{n} + c \tag{5.2.3}$$

where

$$\begin{aligned}
a =& \frac{\omega_t * (1-\alpha) * Workload(wf, InputData)}{ComputingSpeedPerCPU(s) * TimeQuantum} \\
&* \frac{1}{DesiredMoney} * \frac{\sum_{a_i \in CP(SWf)} workload(a_i))}{\sum_{a_j \in CP(wf)} workload(a_j)} \\
b =& \frac{\omega_m * \alpha * Workload(wf, InputData) * MCostPerCPU(s)}{ComputingSpeedPerCPU(s) * DesiredTime} \\
&* \frac{\sum_{a_i \in SWf} workload(a_i))}{\sum_{a_j \in wf} workload(a_j)} \\
c =& (\frac{\omega_m * \alpha * MCostPerCPU(s) * \sum_{a_j \in SWf} workload(a_j)}{DesiredMoney * \sum_{a_i \in wf} workload(a_i)} \\
&+ \frac{\omega_t * (1-\alpha) * \sum_{a_j \in CP(SWf)} workload(a_j)}{DesiredTime * \sum_{a_i \in CP(wf)} workload(a_i)}) \\
&* \frac{Workload(wf, InputData)}{ComputingSpeedPerCPU(s)}
\end{aligned} \tag{5.2.4}$$

Based on Formula 5.2.3, we can calculate a minimal execution cost $Cost_{min}$ and an optimal number of virtual CPUs, *i.e.* $n_{opt}$, according to Formula 5.2.5 and Formula 5.2.6.

$$Cost_{min}(wf, s) = 2 * \sqrt{a * b} + c \tag{5.2.5}$$

$$\boxed{N_{opt} = \sqrt{\frac{b}{a}}} \tag{5.2.6}$$

When the system provisions VMs of $n_{opt}$ virtual CPUs, the cost is the minimal[1] based on Formula 5.2.1, namely $Cost_{min}$, for the execution of Fragment $wf$ at Site $s$.

**Algorithm 1** Single Site VM Provisioning (SSVP)

```
Input: s: the site to execution a fragment; wf: the fragment
    to execute; m: the number of existing virtual CPUs; EVM:
    existing VMs; limit: the maximum number of virtual CPUs
    to instantiate at Site s;
Output: PP: provisioning plan of VMs
begin
1:  $PP \leftarrow \emptyset$
2:  $CPUNumber \leftarrow$ CalculateOptimalNumber($wf, s$)
3:  do
4:      $CurrentCost \leftarrow$ CalculateCost($wf, s, m, EVM, PP$)
5:      $PP' \leftarrow$improve($PP, m, EVM, limit, CPUNumber$)
6:      $Cost \leftarrow$ CalculateCost($wf, s, m, EVM, PP'$)
7:      if $Cost < CurrentCost$ then
8:          $PP \leftarrow PP'$
9:  while $Cost < CurrentCost$
end
```

In order to provision VMs at a site, the system can exploit Algorithm 1 to generate a provisioning plan, which minimizes the cost based on the cost model and $n_{opt}$. In Algorithm 1, Line 2 calculates the optimal number of virtual CPUs to instantiate according to Formulas 5.2.4 and 5.2.6. Since the number of virtual CPUs should be a positive integer, we take $\lceil \sqrt{\frac{b}{a}} \rceil$ as the optimal number of virtual CPUs to instantiate. Lines 4 – 9 optimize the provisioning plan to reduce the cost to execute Fragment $wf$ at Site $s$. Lines 4 and 6 calculate the cost to execute the fragment at the site based on Formulas 5.1.3, 5.1.6 and 5.1.16. Line 5 improves the provisioning plan by inserting

[1] Considering that $a$, $b$ and $n$ are positive numbers, we can calculate the derivative of function 5.2.3 as:

$$ExecutionCost'(N, wf, s) = \frac{d}{dn} ExecutionCost(n, wf, s) = a - \frac{b}{n^2} \tag{5.2.7}$$

When $n$ is smaller than $\sqrt{\frac{b}{a}}$, $ExecutionCost'(n, wf, s)$ is negative and $ExecutionCost(n, wf, s)$ declines when $n$ grows. When $n$ is bigger than $\sqrt{\frac{b}{a}}$, $ExecutionCost'(n, wf, s)$ is positive and $ExecutionCost(n, wf, s)$ increases when $n$ grows. So $ExecutionCost(n, wf, s)$ has a minimum value when $ExecutionCost'(n, wf, s)$ equals zero, *i.e.* $n = \sqrt{\frac{b}{a}}$. And we can calculate the corresponding value of $ExecutionCost'(n, wf, s)$ as shown in Formula 5.2.5.

a new VM, modifying an existing VM or removing an existing VM. If the optimal number of virtual CPUs *CPUNumber* is bigger than the number *ExistingCPUNumber* of virtual CPUs with consideration of current provisioning plan, and existing virtual CPUs, a VM is planed to be inserted in the provisioning plan. The VM is of the type that can reduce the difference between *CPUNumber* and *ExistingCPUNumber*. If *CPUNumber* is smaller than *ExistingCPUNumber*, the difference between *CPUNumber* and *ExistingCPUNumber* is not big and the difference can be reduced by modifying the type of an existing VM, the type of the VM is planed to be modified in the provisioning plan. Otherwise, an existing VM is planed to be removed in the provisioning plan. The VM to be removed is selected among all the existing VMs in order to reduce the most the difference between *CPUNumber* and *ExistingCPUNumber*. If the cost to execute Fragment $wf$ at Site $s$ can be reduced by improving the provisioning plan, the provisioning will be updated (Line 8), and the improvement of provisioning plan continues (Line 9). Note that the direction in the *improve* function of SSVP is determined by comparing *CPUNumber* and *ExistingCPUNumber*, while the function in GraspCC [13] compares the current provisioning plan with all the possible solutions by changing one VM in the provisioning plan, which has no direction, *i.e.* insert, modify or remove, and which is less efficient. While choosing the type of VM to be inserted, modified or removed, storage constraints[1], specifying that the scheduled site should have enough storage resources for executing SWf fragments should be validated. If the storage constraint is not met, more storage resources are planed to be added to the file system of the VM cluster[2] at the site. Note that the number of virtual CPUs to instantiate in the provisioning plan, generated by Algorithm 1, may be smaller than $n_{opt}$ because the cost (time and monetary costs) to initialize the site and to transfer data among different sites is considered.

### *5.3. Cost Estimation*

The cost estimation method is used to estimate the cost to execute a fragment at a site based on the cost model. First, SSVP is used to generate a provisioning plan. Then, the number of virtual CPUs, *i.e.* $n$, and number of VMs to deploy, namely $m$, is known to estimate the time and monetary costs to initiate a site based on Formulas 5.1.7, 5.1.17. In addition, the time and monetary costs to execute the fragment are recalculated using Formulas 5.1.10 and 5.1.19. During scheduling, only available fragments are scheduled. An available fragment indicated that its preceding activities are already scheduled, which means that the location of the input data of the fragment is known. Thus, Formulas 5.1.8 and 5.1.18 are used to estimate the time and monetary costs to transfer the input data of Fragment $wf$ to Site $s$. Afterwards, the total time and monetary costs can be estimated by Formulas 5.1.6 and 5.1.16. Finally, the cost of executing Fragment $wf$ at Site $s$ is estimated based on Formulas 5.1.3, 5.1.4, 5.1.14, 5.1.5, 5.1.15.

## 6. Fragment Scheduling

In this section, we present our approach for fragment scheduling, which is the process of scheduling fragments to sites for execution. First, we present a use case, *i.e.* the SciEvol SWf, which we will use to illustrate fragment scheduling. Then, we explain our approaches for SWf partitioning. Afterwards, we present an adaptation of two state-of-the-art algorithms (LocBased and SGreedy) and our proposed algorithm (ActGreedy).

### *6.1. Use Case*

In this section, in order to illustrate partitioning and scheduling approaches, we present a use case, *i.e.* SciEvol SWf. SciEvol [25] is a SWf for molecular evolution reconstruction that aims at inferring evolutionary relationships, namely to detect positive Darwinian selection, on genome data. It has data and compute intensive activities with data constraints. These characteristics are important to evaluate our scheduling approaches. Figure 2 shows the conceptual structure of SciEvol, which is composed of 13 activities: (1) Multiple Sequence Alignment (MSA) construction; (2) MSA conversion; (3) pre-processing PHYLIP file; (4) tree construction; (5) data construction; (6.1 - 6.6) six evolutionary analysis execution; (7) data construction; (8) data analysis. Let us assume that there are three sites, which are Site 1, Site 2 and Site 3. Activity 1 constructs MSA by formatting the input fasta files stored at site 3. Fasta is a standard type of files in the field of bioinformatics. Activity 2 converts the MSA in fasta format to PHYLIP format. Activity 3 formats the PHYLIP files for evolutionary analysis phase. Activity 4 constructs a phylogenetic tree. Activity 5 is a control activity for preparing the control data for the execution of six evolutionary analysis phases. Activities 6.1 − 6.6 exploit different models to analyze the evolutionary relationships. The corresponding data of the models is stored data at Sites 1, 2 and 3. Activity 7 is a control activity, which gathers the output data of the six evolutionary analysis phases. Activity 8 analyzes the output data of evolutionary analyses to generate statistical readable data for the scientists. Except Activities 6.1 − 6.6, all the other activities are data-intensive. However, the time to process data is small while the time to transfer and store the input or output data takes much time. In Figure 2, “read data” represents that one activity just reads the stored data without modifying it and that the activity should be executed at the corresponding site to read data since the stored data is too big to be moved and can be accessed only within the corresponding site because of configurations, *e.g.* security configuration of a database. The stored data constraints are defined by the following matrix (the activities not listed in the matrix are not fixed).

[1] All the types (A1, A2, A3 and A4) of VMs mentioned in Section 7 can execute the activities of SciEvol in terms of memory.

[2] We assume that a VM cluster exploits a shared file system for fragment execution. In a shared file system, all the computing nodes of the cluster share some data storage that are generally remotely located [20].

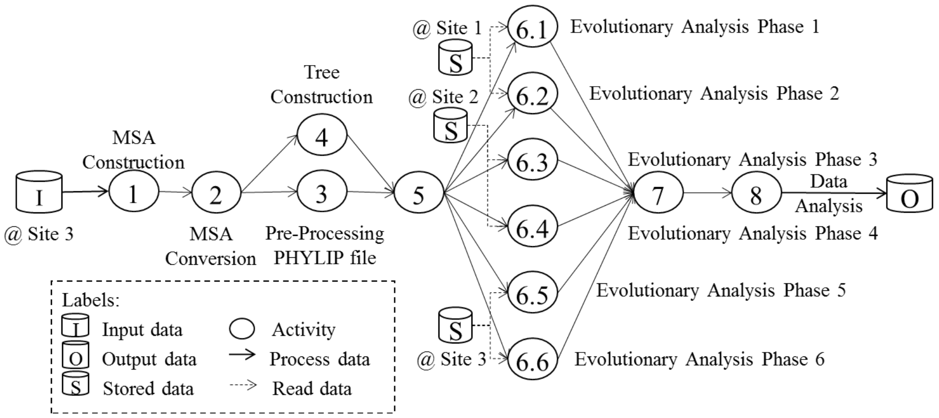


Figure 2: **SciEvol Scientific Workflow.**

| | $s_1$ | $s_2$ | $s_3$ |
|---|---|---|---|
| $a_{6.1}$ | 1 | 0 | 0 |
| $a_{6.2}$ | 1 | 0 | 0 |
| $a_{6.3}$ | 0 | 1 | 0 |
| $a_{6.4}$ | 0 | 1 | 0 |
| $a_{6.5}$ | 0 | 0 | 1 |
| $a_{6.6}$ | 0 | 0 | 1 |

### *6.2. SWf Partitioning*

In this section, we present the algorithms to partition a SWf into fragments. Workflow partitioning is the process of dividing a SWf and input data into several fragments, so that each fragment can be executed at a site. It can be performed by DAG partitioning and data partitioning. DAG partitioning transforms a DAG composed of activities into a DAG composed of fragments while each fragment is a DAG composed of activities and dependencies. Data partitioning divides the input data of a fragment generated by DAG partitioning into several data sets, each of which is encapsulated in a newly generated fragment. This paper focuses on the DAG partitioning.

The smallest granularity of fragment is an activity. Thus, we can encapsulate each activity in one fragment. We call this method activity encapsulation partitioning. The workflow can also be partitioned according to the structure of the SWf. Our previous work [21] proposes a workflow partitioning method that minimizes data transfer among different fragments, *i.e.* data transfer minimization workflow partitioning. Algorithm 2 describes the partitioning method proposed in [21].

**Algorithm 2** Data transfer minimization workflow partitioning

```
Input: $SWf$: a SWf; $S$: a set of sites
Output: $DS$: a set of dependencies to cut in order to partition
    the SWf
begin
 1: $DS \leftarrow \emptyset$
 2: for each $s \in S$ do
 3:     $A \leftarrow$ fixedActvities($SWf, s$)
 4:     $A' \leftarrow$ fixedOutsideUnprocessedActivities($SWf, s$)
 5:     for each $a \in A$ do
 6:         for each $a' \in A'$ do
 7:             $paths \leftarrow$ findPaths($a, a'$)
 8:             for each $path \in paths$ do
 9:                 $ds \leftarrow$ minData($path$)
10:                 $DS \leftarrow DS \bigcup ds$
11: $DS \leftarrow sort(DS)$
12: for each $ds \in DS$ do
13:     if $SWf$ can be partitioned by $DS$ without $ds$) then
14:         $DS \leftarrow DS - ds$
end
```

In Algorithm 2, a set of data dependencies $DS$ are chosen to be removed from the original SWf in order to partition the SWf. In this algorithm, the input data is viewed as a fixed activity. Lines $2 - 10$ select the dependencies to be removed in order to partition the activities of the SWf at each site. Line 4 selects the fixed activities that are outside of Site $s$ and that the corresponding sites are not processed. If one site is processed in the loop of Lines $5 - 11$, it is marked as processed. For the two functions $fixedActivities$ and $fixedOutsideUnprocessedActivities$, each activity is checked to know if the activity is to be selected. Lines $7 - 10$ choose the dependencies to be removed so that the fixed activities at Site $s$ are not connected with the fixed activities at other sites. Line 7 finds all the paths that connect two activities fixed at different sites. A path has a set of data dependencies that can connect two activities without consideration of direction. In order to find all the paths, a depth-first search algorithm can be used. For each path (Line 8), the data dependency that has the least amount of data to be transferred is selected (Line 9) to $DS$ (Line 10). At the same time, the input activity and output activity of the selected data dependency are marked. Line 11 sorts the data dependencies according to the amount of data to be transferred in descending order. If two or more data dependencies have the same amount of data, they are sorted according to the amount of output data of the following activity of each data dependency in descending order. This order allows the activities that have bigger data dependencies with their following activities to be connected with their following activities, after the removing of dependencies (Lines $12 - 14$), in order to reduce the amount of data to be transferred among different fragments. For instance, the order enables that Activity 6.5 and 6.6 are connected with their following activity, *i.e.* Activity 7 in

SciEvol shown in Figure 3. We assume that, in Figures 3, 4 and 5, the order of monetary cost to use VMs in each site is: Site 1 < Site 2 < Site 3. Lines $12 - 14$ remove data dependencies from $DS$ while ensuring that the SWf can always be partitioned with $DS$. Checking if SWf can be partitioned by a set of dependencies can be realized by checking if the corresponding input and output activities of $ds$ are connected with a depth-first search algorithm (Line 13).

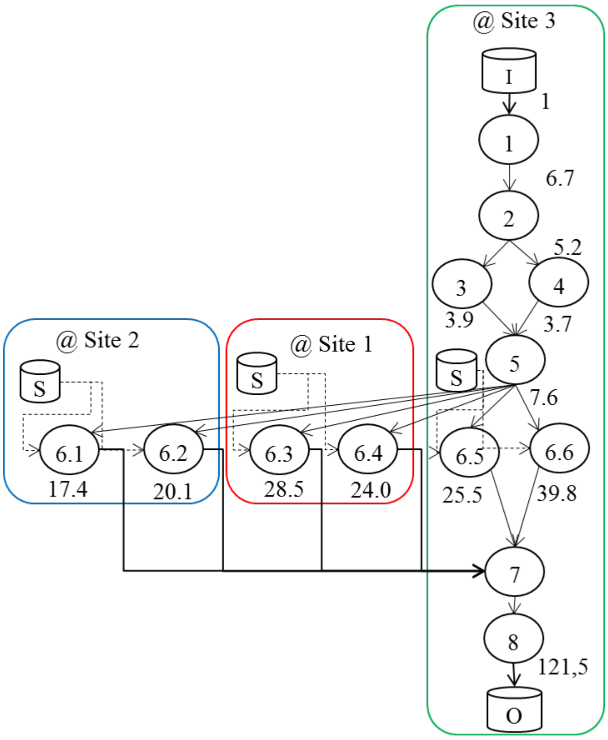


Figure 3: **SWf partitioning and data location based scheduling.** The number represents the relative (compared with the input data) amount of output data for corresponding activities.

### 6.3. *Scheduling approaches*

In this section, we propose three multisite scheduling algorithms. The first one, *LocBased* is adapted from the scheduling algorithm used in our previous work [21], which schedules a fragment to the site that stores the data while reducing data transfer among different sites. The second one, *SGreedy*, is adapted from the greedy scheduling algorithm designed for multi-objective single site scheduling in our previous work [14], which schedules the most suitable fragment to each site. The last one, *ActGreedy*, which combines characteristics of the two adapted algorithms, schedules the most suitable site to each fragment. In addition, we propose that a fixed activity can only be scheduled and executed at the site where the stored data is located. This is applied by analyzing the constraint matrix in all the three algorithms before other steps, which are not explicitly presented in the algorithms.

#### 6.3.1. *Data Location Based Scheduling*

We adapt the scheduling approach proposed in [21] to the multisite cloud environment. Since this approach is based on the location of data, we call it LocBased (data location based) scheduling. First, this algorithm partitions a Fragment $wf$ using the data transfer minimization algorithm (Algorithm 2 in Section 6.2). Then, it schedules the fragment to the data site that stores the required data (*i.e.* stored data for fixed activity) or the biggest part of the input data (for normal activities) in order to reduce the time and monetary costs to transfer data among different sites. However, the granularity of this scheduling algorithm is relatively big and some activities are scheduled at a site that incurs high cost. For instance, the result of this algorithm is shown in Figure 3 while Activity $1, 2, 3, 4, 5, 7$ and $8$ can be scheduled at Site 1, which is less expensive to use VMs than at other sites.

#### 6.3.2. *Site Greedy Scheduling*

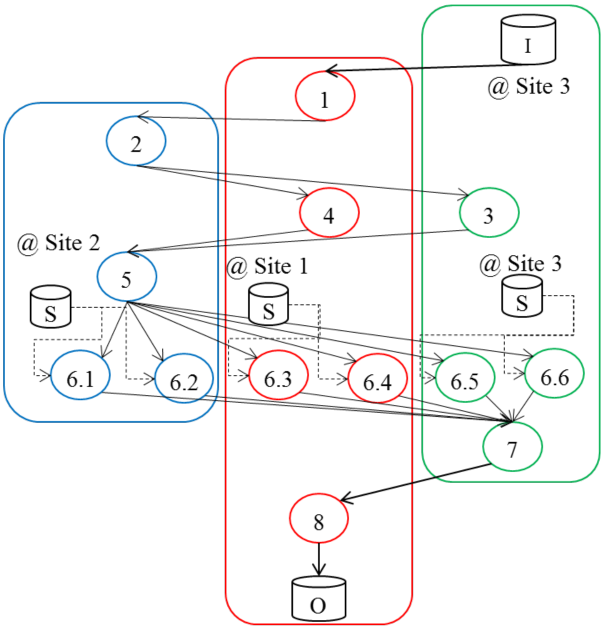


Figure 4: **Site greedy scheduling.**

We adapt a Site Greedy (SGreedy) scheduling algorithm proposed by de Oliveira *et al.* [14], for multiple objectives in a multisite environment. This algorithm intends to keep all the sites busy and chooses the best fragment (not necessarily the optimal one, which could be outside the search space) to allocate for execution at each site. First, this algorithm partitions a SWf according to the activity encapsulation partitioning method. For each site, it schedules the best fragment among available fragments (see Section 5.3). The best fragment takes the minimal cost among all the available fragments for the execution at Site $s$ according to the cost estimation method presented in Section 5.3. The adaptation of the original algorithm to multisite environment is done by replacing scheduling cloud activities to VMs [14] to scheduling fragments to sites with the cost model presented in Section 5.1. However, as shown in Figure 4, this algorithm may break the dataflow between the preceding activity and following activity, which may incur a high cost to transfer the data among different sites, *e.g.* Activities $1, 2, 3, 4, 7, 8$. In addition, in order to avoid useless usage of VMs at Site 1 and Site 2 during the execution of the SWf, the VMs are shut down after finishing the execution of the corresponding activities and restarted for the following activities when the input data is ready. After executing Activities 1, 6.3 and 6.4, the VMs at Site 1 are shut down. The VMs at Site 2 are shut down after executing Activity 2. Since Activity 5 is a control activity, which takes little time to be executed, the VMs at Site 1 and 3 are not shut down after executing Activities 4 and 3. When the execution of Activities 6.5 and 6.6 are to be finished, the VMs at Site 2 are restarted to continue the execution (since the execution of Activities 6.5 and 6.6 may take more time because of big workload). This process may also incur high cost when there are many VMs to restart.

### *6.3.3. Activity Greedy Scheduling*

Based on LocBased and SGreedy, we propose the ActGreedy (Activity Greedy) scheduling algorithm, which is described in Algorithm 3. In this algorithm, all the fragments of a SWf are not scheduled, *i.e.* $Sch(SWf, S) = \emptyset$, at beginning. During the scheduling process, all the fragments are scheduled at a corresponding site, which takes the minimum cost, namely $Sch(SWf, S) = \{Schedule(wf, s) | wf \in SWf, s \in S\}$ and $\forall\ wf \in SWf, \exists\ Schedule(wf, s) \in Sch(SWf, S)$ while cost $Cost(Schedule(wf, s))$ is the minimum compared to schedule Fragment $wf$ to other sites. As a result, the cost $Cost(Sch(SWf, S))$ of executing a SWf in a multisite cloud is minimized. Similar to LocBased, ActGreedy schedules fragments of multiple activities. ActGreedy can schedule a pipeline of activities to reduce data transfer between different fragments, *i.e.* the possible data transfer between different sites. As formally defined in [30], a pipeline is a group of activities with a one-to-one, sequential relationship between them. However, ActGreedy is different from LocBased since it makes a trade-off between time and monetary costs. Similar to SGreedy, ActGreedy schedules the available fragments, while ActGreedy chooses the best site for an available fragment instead of choosing the best fragment for an available site.

**Algorithm 3** Activity greedy scheduling

```
Input: $swf$: a scientific workflow; $S$: a set of sites
Output: $SP$: scheduling plan for $swf$ in $S$
 1: $SP \leftarrow \emptyset$
 2: $SWfCost \leftarrow \infty$
 3: $WF \leftarrow partition(swf)$
 4: do
 5:     $SP' \leftarrow \emptyset$
 6:     $WF \leftarrow Group(WF)$
 7:     do
 8:         $WFA \leftarrow GetAvailableFragments(WF, SP')$
 9:         if $WFA \neq \emptyset$ then
10:             for each $wf \in WFA$ do
11:                 $s_{opt} \leftarrow BestSite(wf, S)$
12:                 $SP' \leftarrow SP' \bigcup \{Schedule(wf, s_{opt})\}$
13:                 update $CurrentSWfCost$
14:     while not all the fragments $\in WF$ are scheduled
15:     if $CurrentSWfCost < SWfCost$ then
16:         $SP \leftarrow SP'$
17:         $SWfCost \leftarrow CurrentSWfCost$
18: while $CurrentSWfCost < SWfCost$
end
```

ActGreedy chooses the best site for each fragment. First, it partitions a SWf according to the activity encapsulation partitioning method (Line 3). Then, it groups the fragments of three types into bigger fragments to be scheduled (Line 6). The first type is a pipeline of activities. We use a recursive algorithm presented in [30] to find pipelines. Then, the fragments of corresponding activities of each pipeline are grouped into a fragment. If there are stored activities of different sites in a fragment of a pipeline, the fragment is partitioned into several fragments by the data transfer minimization algorithm (Algorithm 2) in the *Group* function. The second type is the control activities. If it has only one preceding activity, a control activity is grouped into the fragment of its preceding activity. If it has multiple preceding activities and only one following activity, a control activity (Activity 7) is grouped into the fragment of its following activity (Activity 8). If it has multiple preceding activities and multiple following activities, a control activity (Activity 5) is grouped into the fragment of one of its preceding activities (Activity 3), which has the most data dependencies among all its preceding activities, *i.e.* the amount of data to be transferred in the data dependency is the biggest. It reduces data transfer among different fragments, namely the data transfer among different sites, to group the fragments for pipelines and control activities. The third type is the activities that are scheduled at the same site and that they have dependencies to connect each activity of them. Afterwards, Line 8 gets the available fragments (see Section 5.3) to be scheduled to the best site (Line $11 - 12$), which takes the minimal cost among all the sites to execute the fragment. The cost is estimated according to the method presented in Section 5.3. When estimating the cost, if the scheduled fragment has data dependencies with fixed activities, the cost to transfer the data in these data dependencies will be taken into consideration. The loop (Lines $7 - 14$) schedules each fragment to the best site while the big loop (Lines $4 - 18$) improves the scheduling plans by rescheduling the fragments after grouping the fragments at the same site, which ensures that the final scheduling plan corresponds to smaller cost to execute a SWf.

As shown in Figure 5, this algorithm has relatively small granularity compared with LocBased. ActGreedy exploits data location information to select the best site in order to make a trade-off between the cost for transferring data among different sites and another cost, *i.e.* the cost to provision the VMs and the cost to execute fragments. Figure 5 shows the scheduling result. If the amount of data increases and the desired execution time is small, Activity 7 and Activity 8 may be scheduled at Site 3, which takes less cost to transfer data. In order to avoid useless usage of VMs, the VMs at Site 1 are shut down when the site

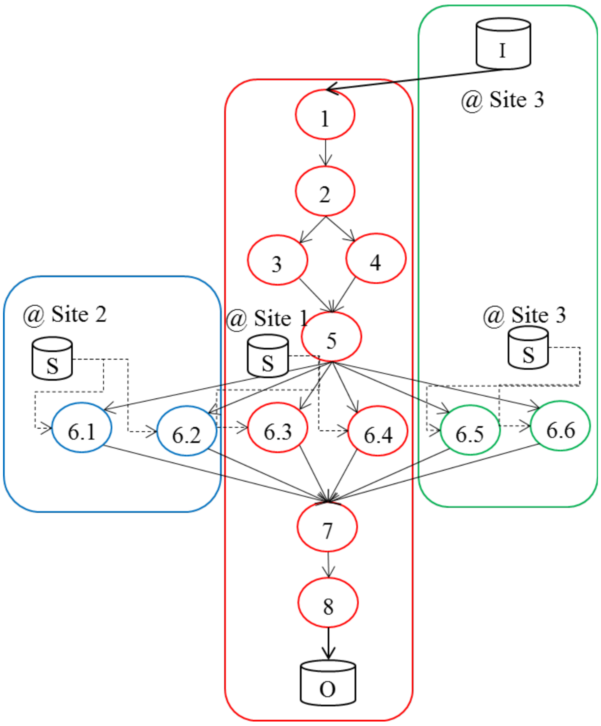


Figure 5: **Activity greedy scheduling.**

is waiting for the output data of the execution of Site 3, namely the execution of Activity 6.5 and Activity 6.6.

### 6.3.4. Solution analysis

Let us assume that we have $n$ activities and $s$ cloud sites and $f$ fixed activities. The solution search space of a general scheduling problem is $O(s^n)$. The solution search space of a scheduling problem after fixing the activities becomes $O(s^{n-f})$. Even though the search space is reduced because of *stored data constraints*, the problem remains hard since the search space exponentially increases when $n$ becomes bigger. For instance, assuming that we have a SWf with 77 activities (6 fixed activities) to be schedule at 3 sites, the search space of a general scheduling problem is $O(3^{77})$, *i.e.* $O(5.47 * 10^{36})$, and that of the scheduling problem with fixed activities is $O(3^{71})$, *i.e.* $O(7.51 * 10^{33})$. Some input or output activities may be related to the fixed activities, but they are free to be scheduled at any site. In general, the number of fixed activities is quite small compared with the number of other activities. The complexity of our proposed algorithm (ActGreedy) is $O(s * (n - f))$, which is much smaller than $O(s^{n-f})$. As a result, our solution can resolve the problem within reasonable scheduling time, *i.e.* the time to generate scheduling plans.

The knowledge of the location of stored data can be obtained by the metadata of files stored at each site, which is easy to get before SWf execution. Then, the knowledge of fixed activities can be generated with the dependencies between activities and data. Thus, knowing fixed activities is not a problem.

Our solution generates a scheduling plan that corresponds to the minimum cost to execute a SWf in a multisite cloud since all the fragments are scheduled to a site, which takes the minimum or near-minimum cost to execute them. The fragments of small granularity are scheduled to the best site, which takes the minimum cost to execute the fragments, by the small loop (Lines 7 – 14) while the scheduling of fragments of big granularity, *i.e.* the activities of a site, is ensured by big loop (Lines 4 – 18). Thus, our proposed algorithm can generate a scheduling plan which may minimize the cost. Since the weight of each objective is positive and the generated scheduling plan may minimize the sum function of multiple objectives, the solution may also be Pareto optimal [38] [24]. Although, in some rare cases, *e.g.* the cost to transfer data between different sites affects the scheduling plans, the cost corresponding to the generated scheduling plan is not minimum, our proposed solution generates a near-optimal scheduling plan. Since the experiments presented in this paper are not rare cases, and that the scheduling plan generated by our algorithm is already Pareto optimal (no better scheduling plan can be found by estimating the cost of other scheduling plans), we do not compare it with another optimal solution, which may not even exist.

Note that the proposed algorithm and the results shown in Section 7 are sensitive to the cost model. Although the cost model is mentioned in previous work [14], it is not used in a multisite environment with stored data constraints. We evaluated all the cost model components in our previous work, which indicates that it is a suitable model to work with for multi-objective scheduling.

## 7. Experimental Evaluation

In this section, we present an experimental evaluation of the VM provisioning plan generation (SSVP) and fragment scheduling (ActGreedy) algorithms. All experiments are based on the execution of the SciEvol SWf in Microsoft Azure multisite cloud. First, we compare SSVP with GraspCC. Then we compare ActGreedy with LocBased and SGreedy, as well as with two general algorithms, *i.e.* Genetic and Brute-force. In the experiments, we consider three Azure [2] sites to execute SciEvol SWf, namely West Europe as Site 1, Japan West as Site 2, Japan East as Site 3. During the experiments, the the life circle of VM is composed of creation, start, configuration, stop and deletion. The creation, start, stop and deletion of a VM is managed by using Azure CLI [3]. The configuration of VM is realized by Linux SSH command. In the experiments, workflow partitioner, multisite scheduler and single site initialization are simulated, but the execution of fragments is performed in a real environment by Chiron [26], which is a SWfMS to support the execution of SWfs using an algebraic approach. We conduct the experiments with two goals. The first one is to show that SSVP is suitable to dynamic provisioning of VMs by making a good trade-off among different objectives for the execution of fragments. The second goal is to show that ActGreedy takes the smallest cost to execute a SWf in a multisite cloud within reasonable time by making a trade-off of different objectives based on SSVP. Microsoft Azure provides 5 tiers of VM, which are basic tier, standard tier, optimized compute, performance optimized compute and compute intensive. Each tier of VM

Table 2: **Parameters of different types of VMs.** Type represents the type of VMs. vCPUs represents the number of virtual CPUs in a VM. RAM represents the size of memory in a VM. Disk represents the size of hard disk in a VM. CC represents the computing capacity of VMs. MC represents Monetary Cost.

| Type | vCPUs | RAM | Disk | CC | MC @ WE | MC @ JW | MC @ JE |
|---|---|---|---|---|---|---|---|
| A1 | 1 | 1.75 | 70 | 9.6 | 0.0447 | 0.0544 | 0.0604 |
| A2 | 2 | 3.5 | 135 | 19.2 | 0.0894 | 0.1088 | 0.1208 |
| A3 | 4 | 7 | 285 | 38.4 | 0.1788 | 0.2176 | 0.2416 |
| A4 | 8 | 14 | 605 | 76.8 | 0.3576 | 0.4352 | 0.4832 |

contains several types of VMs. In one Web domain, users can provision different types of VMs at the same tier. In our experiments, we consider 4 types, namely $A1$, $A2$, $A3$ and $A4$, in the standard tier. The features of the VM types are summarized in Table 2. In Azure, the time quantum is one minute. In addition, the average time to provision a VM is estimated as 2.9 minutes. Each VM uses Linux Ubuntu 12.04 (64-bit), and is configured with the necessary software for SciEvol. All VMs are configured to be accessed using Secure Shell (SSH).

Table 3: **Workload Estimation.**

| Activity | Number of Fasta Files | | |
|---|---|---|---|
| | 100 | 500 | 1000 |
| | Estimated Workload (in GFLOP) | | |
| 1 | 1440 | 10416 | 20833 |
| 2 | 384 | 2778 | 5556 |
| 3 | 576 | 4167 | 8333 |
| 4 | 1440 | 10416 | 20833 |
| 6.1 | 5760 | 41667 | 83334 |
| 6.2 | 10560 | 76389 | 152778 |
| 6.3 | 49920 | 361111 | 722222 |
| 6.4 | 59520 | 430556 | 861111 |
| 6.5 | 75840 | 548611 | 1097222 |
| 6.6 | 202560 | 1465278 | 2930556 |
| 8 | 6720 | 48611 | 97222 |

In the experiments, we use 100, 500, 1000 fasta files generated from the data stored in a genome database [7][8]. The programs used are: mafft (version 7.221) for Activity 1, ReadSeq 2.1.26 for Activity 2, raxmhpc (7.2.8 alpha) for Activity 4, pamlX1.3.1 for Activities 6.1 – 6.6, in-house script for Activity 3 and Activity 8, and Activity 5 and Activity 7 exploit a PostgreSQL database management system to process data. The percentage of the workload, *i.e.* $\alpha$ in Formula 5.1.10, that can be parallelized is 96.43%. In addition, the input data of the SWf is stored at a data server of Site 3, which is accessible to all the sites in the cloud using *SCP* command (a Linux command). The estimated workload (in GFLOP) of each activity of SciEvol SWf for different numbers of input fasta files is shown in Table 3 and the estimated amount of data transferred in each dependency of SciEvol SWf is listed in Table 4. In Table 4, $e_{i,j}$ represents the data dependency between Activity $i$ and Activity $j$ while Activity $j$ consumes the output data of Activity $i$. Let DataSize($e_{i,j}$) represent the estimated amount of data in dependency $e_{i,j}$. Then, we have DataSize($e_{2,3}$) = DataSize($e_{2,4}$); DataSize($e_{4,5}$) = DataSize($e_{3,5}$); DataSize($e_{5,6.1}$) = DataSize($e_{5,6.2}$) = DataSize($e_{5,6.3}$) = DataSize($e_{5,6.4}$) = DataSize($e_{5,6.5}$) = DataSize($e_{5,6.6}$).

Table 4: **Estimated amount of data transferred in a dependency.** Input data represents the number of input fasta files for executing SciEvol SWf.

| Dependency | Number of Fasta Files | | |
|---|---|---|---|
| | 100 | 500 | 1000 |
| | Estimated Amount of Data | | |
| Input Data | 1 | 5 | 10 |
| $e_{1,2}$ | 6 | 32 | 67 |
| $e_{2,3}$ | 5 | 24 | 52 |
| $e_{3,5}$ | 3 | 17 | 39 |
| $e_{4,5}$ | 3 | 16 | 37 |
| $e_{5,6.1}$ | 6 | 33 | 76 |
| $e_{6.1,7}$ | 16 | 85 | 174 |
| $e_{6.2,7}$ | 20 | 100 | 201 |
| $e_{6.3,7}$ | 28 | 140 | 285 |
| $e_{6.4,7}$ | 23 | 118 | 240 |
| $e_{6.5,7}$ | 24 | 125 | 255 |
| $e_{6.6,7}$ | 34 | 175 | 348 |
| $e_{7,8}$ | 120 | 605 | 1215 |

In the tables and figures, the unit of time is minute, the unit of monetary cost is Euro, the unit of RAM and Disk is Gigabytes, the unit of data is MegaByte (MB), the computing capacity of VMs is GigaFLOPS (GFLOPS) and the unit of workload is GigaFLOP (GFLOP). $\omega_t$ represents the weight of time cost. $A1$, $A2$, $A3$ and $A4$ represent the types of VMs in Azure. [Type of VM] * [number] represents provisioning [number] of VMs of [Type of VM] type, *e.g.* A1 * 1 represents provisioning one VM of A1 type. WE represents West Europe; JW Japan West and JE Japan East. The cost corresponds to the price in Euro of Azure on July 27, 2015.

### *7.1. VM Provisioning*

This section presents the experimental results to compare the performance of SSVP over GraspCC in two aspects. The first aspect is that SSVP can estimate cost more accurately based on our proposed cost model than GraspCC. The second aspect is that the provisioning plans generated by SSVP incur less cost than that generated by GraspCC. In this section, the cost is for the execution of fragments, which is calculated based on Formulas 5.1.3, 5.1.4, 5.1.14, 5.1.5, 5.1.15 while the time and monetary costs are obtained from the real execution with the desired time, desired monetary cost and estimated workload configured by a user.

We execute a fragment (Activities 1, 2, 3, 4, 5, 6.5 and 6.6 of SciEvol) with 100 fasta files, at Site 3 for different weights of

Table 5: **VM Provisioning Results.**

| Algorithm | | SSVP | | | GraspCC | | |
|---|---|---|---|---|---|---|---|
| $\omega_t$ | | 0.1 | 0.5 | 0.9 | 0.1 | 0.5 | 0.9 |
| Provisioning Plan | | $A3 * 1$ | $A4 * 1$ | $A4 * 3$ | $A1 * 6$ | $A2 * 3$ | |
| Estimated | Execution Time | 95 | 55 | 34 | 60 | | |
| | Monetary Cost | 0.38 | 0.44 | 0.75 | 0.36 | | |
| | Cost | 1.3094 | 1.1981 | 0.7631 | 1.1882 | 1.104 | 1.0208 |
| Real | Execution Time | 98 | 54 | 35 | 113 | 100 | |
| | Monetary Cost | 0.40 | 0.43 | 0.81 | 0.64 | 0.60 | |
| | Cost | 1.3472 | 1.1748 | 0.7879 | 2.1181 | 1.8199 | 1.6973 |

execution time and monetary costs. We assume that the limitation of the number of virtual CPUs is 32. The estimated workload of this fragment is 192,000 GFLOP. The desired execution time is set as 60 minutes and the maximum execution time is defined as 120 minutes. The desired monetary cost is configured as 0.3 Euros and the maximum monetary cost is 0.6 Euros. The deployment plans presented in Table 5 are respectively generated by SSVP, and GraspCC [13]. Table 5 shows the result of the experiments to execute the fragment with different weights of execution time and monetary costs. The execution time, monetary cost and cost is composed of the time or the cost of VM provisioning and fragment execution. For SSVP, the difference between estimated and real execution time ranges from 1.9% to 3.1%, the difference for monetary cost ranges from 2.0% to 6.5% and the difference for the cost is between 2.0% to 3.2%. In fact, the difference between the estimated and real values also depends on the parameters configured by the users. Table 5 shows that SSVP can make an acceptable estimation based on different weights of objectives, *i.e.* time and monetary costs.

GraspCC is based on three strong assumptions. The first assumption is a single site, *i.e.* GraspCC is not designed for multisite clouds. The second assumption is that the entire workload of each activity can be executed in parallel, which may not be realistic since many activities cannot be parallelized. The third assumption is that more VMs can reduce execution time without any bad impact on the cost, *e.g.* higher monetary cost, for the whole execution of a fragment. These three assumptions lead to inaccuracies of the estimation of execution time and monetary costs. In addition, as it is only designed for the time quantum of one hour, GraspCC always generates a provisioning plan that contains the possible largest number of virtual CPUs to reduce the execution time to one time quantum, namely one hour. In Azure, since the time quantum is one minute, it is almost impossible to reduce the execution time to one time quantum, namely one minute. In order to use GraspCC in Azure, we take the time quantum of one hour for GraspCC. GraspCC does not take into consideration the cost (time and monetary costs) to provision VMs, which also brings inaccuracy to the estimated time. Moreover, GraspCC is not sensitive to different values of weight, which are $\omega_t$ and $\omega_m$. But SSVP is sensitive to different values of weight because of using the optimal number of virtual CPUs calculated based on the cost model. The final provisioning plan of GraspCC is listed in Table 5. GraspCC generates the same provisioning plan for different values of $\omega_t$ (0.5 and 0.9). In addition, the difference between the estimated time and the real time is 88.3% ($\omega_t = 0.1$) and 66.7% ($\omega_t = 0.5$ and $\omega_t = 0.9$). However, the difference corresponding to the cost model of SSVP is under 3.1%. Finally, compared with SSVP, the corresponding real cost of the GrapsCC algorithm is 57.2% ($\omega_t = 0.1$), 54.9% ($\omega_t = 0.5$) and 115.4% ($\omega_t = 0.9$) bigger.

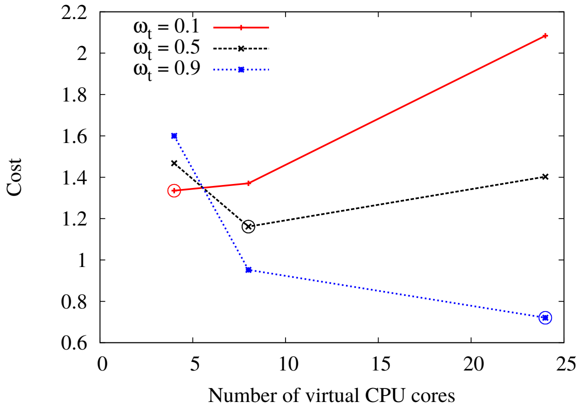


Figure 6: **Cost for different values of $\omega_t$ with three provisioning plans.** The circled points represents the number of virtual CPUs corresponding to the provisioning plan generated by SSVP and the corresponding cost of the execution of fragment.

Figure 6 shows the cost for different provisioning plans and different weights of execution time and monetary costs. According to the provisioning plan generated by SSVP, 4, 8 and 24 virtual CPUs are instantiated when $\omega_t$ is 0.1, 0.5 and 0.9. The corresponding cost is the minimum value in each polyline. The three polylines show that SSVP can generate a good VM provisioning plan, which reduces the cost based on the cost model. The differences between the highest cost and the cost of corresponding good provisioning plans are: 56.1% ($\omega_t = 0.1$), 26.4% ($\omega_t = 0.5$) and 122.1% ($\omega_t = 0.9$).

We also execute SciEvol with 500 and 1000 fasta files at site 3. The setup parameters are listed in Table 6 and the results are shown in Table 7. Since it needs bigger computing capacity to process more input fasta files, we increase the limitation of the number of virtual CPUs, *i.e.* 64 virtual CPUs for 500 fasta files and 128 virtual CPUs for 1000 fasta files. From the tables, we can see that as the estimated workload and desired monetary cost of the SWf grow, more virtual CPUs are planed to be de-

Table 6: **Setup Parameters.** "Number" represents the number of input fasta files. "Limit" represents the maximal number of virtual CPUs that can be instantiated in the cloud. Maximum values are twice the desired values.

| Number | | 500 | 1000 |
|---|---|---|---|
| Desired | Execution Time | 60 | 60 |
| | Monetary Cost | 2 | 6 |
| Maximum | Execution Time | 120 | 120 |
| | Monetary Cost | 4 | 12 |
| Limit | | 64 | 128 |
| Estimated Workload | | $1,401,600$ | $2,803,200$ |

ployed at the site. SSVP generates different provisioning plans for each weight of time cost. However, for the same number of input fasta files, GraspCC generates the same provisioning plan for different weights of time cost, namely $A1 * 1$, $A3 * 10$ for 500 fasta files and $A2 * 1$, $A4 * 10$ for 1000 fasta files. The execution time corresponding to both SSVP and GraspCC, exceeds the maximum execution time. However, SSVP has some important advantages, *e.g.* precise estimation of execution time and smaller corresponding cost.

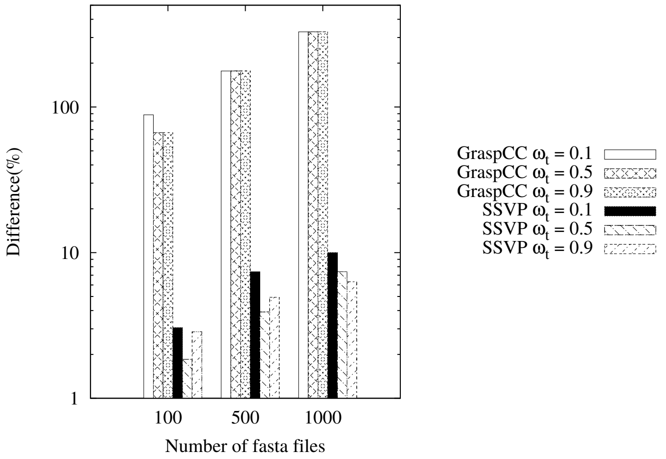


Figure 7: **Difference between estimated time and real time.**

The difference between estimated time and real time is calculated based on Formula 7.1.1. As shown in Figure 7, the difference between estimated execution time and real execution time corresponding to GraspCC is much higher than that corresponding to the cost model of SSVP, which ranges between 66.7% and 328.3%. As the number of fasta files increases, the difference goes up, *i.e.* it is more difficult to estimate the time. However, the difference corresponding to the cost model of SSVP is always under 11%.

$$Difference = \frac{EstimatedTime - RealTime}{RealTime} * 100\% \quad (7.1.1)$$

The cost corresponding to different numbers of fasta files are shown in Figure 8. It can be seen from Figure 8(a), Figure 8(b) and Figure 8(c) that the cost corresponding to GraspCC is always higher than that corresponding to SSVP with different amounts of input data because SSVP is based on a more accurate cost model and is designed for the quantum of one minute. Based on Formula 7.1.2, compared with SSVP, the cost corresponding to GraspCC is up to 115.4% higher. The cost for GraspCC is a line in Figures 8(b) and 8(c), since GraspCC is not sensitive to the weights of time cost and it generates the same VM provisioning plans, the cost of which is a line. However, since SSVP is sensitive to different values of the weights of execution time, it can reduce the cost at large.

$$Difference = \frac{Cost(GraspCC) - Cost(SSVP)}{Cost(SSVP)} * 100\% \quad (7.1.2)$$

From the experimental results, we can get the conclusion that SSVP can generate better VM provisioning plans than GraspCC because of accurate cost estimation of the cost model.

### *7.2. Scheduling Approaches*

Table 8: **Setup parameters.** "Number" represents the number of input fasta files. "Limit" represents the maximal number of virtual CPUs that can be instantiated in the cloud. "DET" represents Desired Execution Time and "DMC" represents Desired Monetary Cost.

| Number | 100 | 500 | 1000 |
|---|---|---|---|
| Limit | 350 | 350 | 350 |
| Estimated workload | $414,720$ | $3,000,000$ | $6,000,000$ |
| DET | 60 | 60 | 60 |
| DMC | 0.3 | 3 | 6 |

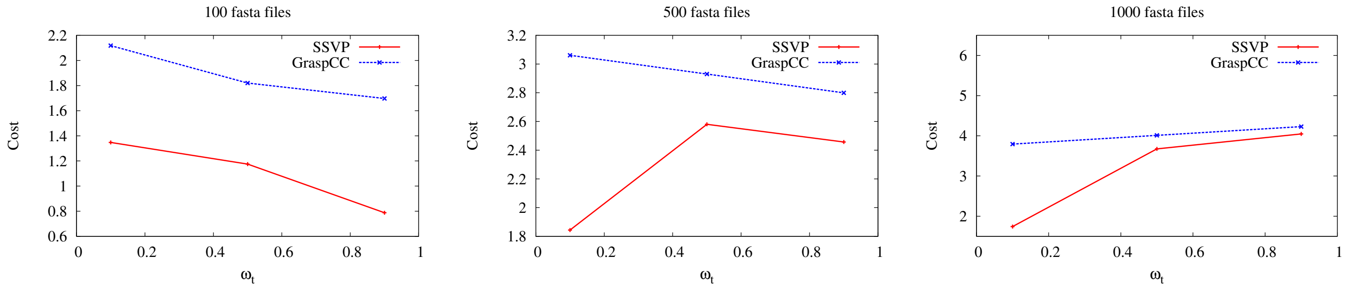
Table 9: **VM Provisioning Plans (100 fasta files).**

| Algorithm | Site | $\omega_t$ | | |
|---|---|---|---|---|
| | | 0.1 | 0.5 | 0.9 |
| LocBased | WE | $A3 * 1$ | $A4 * 1$ | $A4 * 2$ |
| | JW | $A3 * 1$ | $A4 * 1$ | $A4 * 1$ |
| | JE | $A1 * 1, A2 * 1$ | $A4 * 1$ | $A4 * 3$ |
| SGreedy | WE | $A3 * 1$ | $A4 * 1$ | $A4 * 2$ |
| | JW | $A2 * 1, A3 * 1$ | $A4 * 1$ | $A4 * 1$ |
| | JE | $A1 * 1, A2 * 1$ | $A4 * 1$ | $A4 * 3$ |
| ActGreedy | WE | $A2 * 1$ | $A2 * 1; A3 * 1$ | $A4 * 2$ |
| | JW | $A3 * 1$ | $A4 * 1$ | $A4 * 1$ |
| | JE | $A1 * 1, A2 * 1$ | $A4 * 1$ | $A4 * 2$ |

In this section, we present the experimental results to show that our proposed scheduling algorithm, *i.e.* ActGreedy, leads to the least cost for the execution of SWf in a multisite cloud environment. We schedule the fixed activities at the site where the data is stored and use the three scheduling algorithms, namely LocBased, SGreedy and ActGreedy, to schedule other activities of SciEvol at the three sites. In addition, we implemented a genetic algorithm and a brute-force algorithm that generate the best scheduling plans similar to those generated by ActGreedy. Brute-force measures the cost of all the possible scheduling plans and finds the optimal one, corresponding to the minimum cost to execute SWfs in a multisite cloud. The principle of a genetic algorithm [35] is to encode possible scheduling plans into

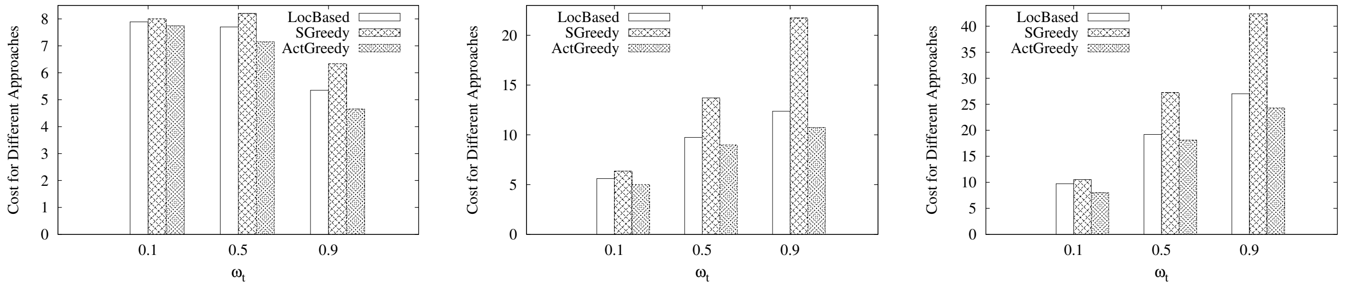
Table 7: **VM Provisioning Results.** "Number" represents the number of input fasta files. The provisioning plan represents the plan generated by the corresponding algorithms.

| Number | | | 500 | | | 1000 | | |
|---|---|---|---|---|---|---|---|---|
| $\omega_t$ | | | 0.1 | 0.5 | 0.9 | 0.1 | 0.5 | 0.9 |
| SSVP | Provisioning Plan | | $A2 * 1, A4 * 1$ | $A4 * 3$ | $A4 * 7$ | $A4 * 2$ | $A4 * 6$ | $A4 * 11$ |
| | Estimated | Execution Time | 328 | 194 | 150 | 473 | 290 | 260 |
| | | Monetary Cost | 3.29 | 4.60 | 7.93 | 7.59 | 13.62 | 21.70 |
| | | Cost | 2.0263 | 2.7640 | 2.6419 | 1.9271 | 3.5462 | 4.2602 |
| | Real | Execution Time | 299 | 177 | 136 | 424 | 294 | 244 |
| | | Monetary Cost | 2.99 | 4.42 | 8.34 | 6.90 | 14.71 | 23.21 |
| | | Cost | 1.8438 | 2.5800 | 2.4572 | 1.7417 | 3.6758 | 4.0468 |
| GraspCC | Provisioning Plan | | $A1 * 1, A3 * 10$ | | | $A2 * 1, A4 * 10$ | | |
| | Estimated | Execution Time | 60 | | | 60 | | |
| | | Monetary Cost | 2.48 | | | 4.95 | | |
| | | Cost | 1.2144 | 1.1191 | 1.0238 | 0.8429 | 0.9127 | 0.9825 |
| | Real | Execution Time | 166 | | | 257 | | |
| | | Monetary Cost | 6.19 | | | 22.43 | | |
| | | Cost | 3.06 | 2.93 | 2.80 | 3.79 | 4.01 | 4.23 |



(a) **Cost for 100 fasta files.** (b) **Cost for 500 fasta files.** (c) **Cost for 1000 fasta files.**

Figure 8: **Cost for different number of fasta files.**



(a) **Comparision for 100 fasta files.** (b) **Comparision for 500 fasta files.** (c) **Comparision for 1000 fasta files.**

Figure 9: **Cost for different scheduling algorithms.** The cost is calculated according to Formula 5.1.2.

Table 10: **VM Provisioning Plans (500 fasta files).**

| Algorithm | Site | $\omega_t$ | | |
|---|---|---|---|---|
| | | 0.1 | 0.5 | 0.9 |
| LocBased | WE | $A3 * 1, A4 * 1$ | $A4 * 4$ | $A4 * 7$ |
| | JW | $A4 * 1$ | $A4 * 2$ | $A4 * 3$ |
| | JE | $A1 * 1, A4 * 1$ | $A4 * 3, A3 * 1$ | $A4 * 8$ |
| SGreedy | WE | $A3 * 1, A4 * 1$ | $A4 * 4$ | $A4 * 7$ |
| | JW | $A4 * 1$ | $A4 * 2$ | $A4 * 3$ |
| | JE | $A1 * 1, A3 * 1$ | $A3 * 1, A4 * 3$ | $A4 * 8$ |
| ActGreedy | WE | $A4 * 1$ | $A2 * 1; A4 * 2$ | $A4 * 5$ |
| | JW | $A4 * 1$ | $A4 * 2$ | $A4 * 3$ |
| | JE | $A1 * 1, A4 * 1$ | $A3 * 1, A4 * 3$ | $A4 * 9$ |

Table 11: **VM Provisioning Plans (1000 fasta files).**

| Algorithm | Site | $\omega_t$ | | |
|---|---|---|---|---|
| | | 0.1 | 0.5 | 0.9 |
| LocBased | WE | $A2 * 1, A4 * 1$ | $A4 * 6$ | $A4 * 9$ |
| | JW | $A4 * 2$ | $A4 * 3$ | $A4 * 4$ |
| | JE | $A2 * 1, A3 * 1, A4 * 1$ | $A4 * 5$ | $A4 * 11$ |
| SGreedy | WE | $A4 * 2$ | $A4 * 6$ | $A4 * 10$ |
| | JW | $A4 * 2$ | $A4 * 3$ | $A4 * 4$ |
| | JE | $A2 * 1, A3 * 1, A4 * 1$ | $A4 * 5$ | $A4 * 11$ |
| ActGreedy | WE | $A2 * 1, A4 * 1$ | $A4 * 4$ | $A4 * 8$ |
| | JW | $A4 * 2$ | $A4 * 3$ | $A4 * 4$ |
| | JE | $A2 * 1, A3 * 1, A4 * 1$ | $A4 * 6$ | $A4 * 12$ |

a population of chromosomes, and subsequently to transform the population using standard operations of selection, crossover and mutation, producing successive generations, until the convergence condition is met. In the experiments, we set the convergence condition so that the cost of scheduling plans should be equal or smaller than that generated by ActGreedy. We use 100 chromosomes and set the number of generations as 1 for the experiments of different numbers of input files and different values of $\alpha$. We choose a random point for the crossover and mutation operation. The experimental results[1] are shown in Figure 9. The setup parameters are shown in Table 8 and provisioning plans, which are generated by SSVP, are listed in Table 9, Table 10 and Table 11. We assume that the data transfer rate between different sites is 2MB/s. The monetary cost to transfer data from Site 1 to other sites is 0.0734 Euros/GB and the monetary cost for Site 2 and Site 3 is 0.1164 Euros/GB [5]. In the experiments, the critical path of SciEvol SWf is composed of Activities 1, 2, 4, 5, 6.6, 7, 8.

LocBased is optimized for reducing data transfer among different sites. The scheduling plan generated by this algorithm is shown in Figure 3. However, the different monetary costs of the three sites are not taken into consideration. In addition, this algorithm directly schedules a fragment, which contains multiple activities. Some activities are scheduled at a site, which is more expensive to use VMs, *e.g.* Site 3. As a consequence, the scheduling plan may correspond to higher cost. SGreedy schedules a site to the available activity, which takes the least cost. The corresponding scheduling plan is shown in Figure 4. SGreedy does not take data location into consideration and may schedule two continuous activities, *i.e.* one preceding activity and one following activity, to two different sites, which takes time to transfer data and to provision VMs. As a result, this algorithm may lead to higher cost. ActGreedy can schedule each fragment to a site that takes the least cost to execute it, which leads to smaller cost compared with LocBased and SGreedy. In addition, ActGreedy can make adaptive modification for different numbers of input fasta files. For instance, there are three situations where Activity 7 and Activity 8 are scheduled at Site 3 to reduce the cost of data transfer while the other scheduling plans are the same as shown in Figure 8(c). The three situations are when there are 500 input fasta files and $\omega_t = 0.9$ and when there are 1000 input fasta files and $\omega_t = 0.5$ or $\omega_t = 0.9$.

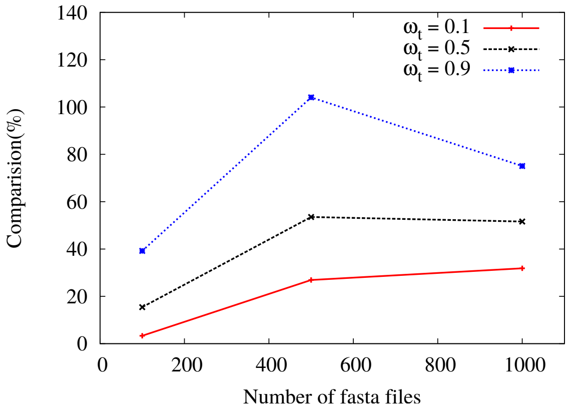


Figure 10: **Comparision of cost between SGreedy and ActGreedy for different number of input fasta files.** The cost is calculated according to Formula 5.1.2.

First, we analyze the cost based on Formula 5.1.2 and Formula 5.1.3. The time and monetary costs to execute a fragment at a site are measured during execution. Both the time and monetary costs are composed of three parts, *i.e.* site execution, data transfer and fragment execution. Based on Formulas 5.1.3, 5.1.4, 5.1.14, 5.1.5, 5.1.15, the cost to execute a fragment is calculated. Based on Formula 5.1.2 and Formula 5.1.3, the cost to execute a SWf is calculated. The cost corresponding to 100 fasta files is shown in Figure 9(a). In order to execute SciEvol SWf with 100 fasta files, ActGreedy can reduce 1.85% ($\omega_t$ = 0.1), 7.13% ($\omega_t$ = 0.5) and 13.07% ($\omega_t$ = 0.9) of the cost compared with LocBased. Compared with SGreedy, ActGreedy can reduce 3.22% ($\omega_t$ = 0.1), 12.80% ($\omega_t$ = 0.5) and 26.60% ($\omega_t$ = 0.9) of the cost. Figure 9(b) shows the cost for different values of $\omega_t$ for processing 500 fasta files. The experimental results show that ActGreedy is up to 13.15% ($\omega_t$ = 0.9) better than LocBased and up to 50.57% ($\omega_t$ = 0.9) better than SGreedy for 500 fasta files. In addition, Figure 9(c) shows the experimental results for 1000 fasta files. The results show that LocBased takes up to 21.75% ($\omega_t$ = 0.1) higher cost than ActGreedy and that SGreedy takes up to 74.51% ($\omega_t$ = 0.9) higher

[1] In the experiments, in order to facilitate the data transfer of many small files, we use the *tar* command to archive the small files into one big file before data transferring.

cost than ActGreedy when processing 1000 fasta files. Figure 10 describes the difference between the worst case (SGreedy) and the best case (ActGreedy). In Figure 10, the $Y$ axis represents the advantage[1] of ActGreedy compared with SGreedy. It can be seen from Figure 10 that ActGreedy outperforms SGreedy and its advantage becomes obvious when $\omega_t$ grows. When there are more input fasta files, the advantage is bigger at first. But it decreases when the number of fasta files grows from 500 to 1000 since the cost corresponding to ActGreedy increases faster.

Accordingly, the cost calculated according to Formula 5.1.1 is shown in Figure 11. In the real execution, the time and monetary costs for the whole execution of a SWf are measured and the real cost can be calculated by Formula 5.1.1. From Figures 11(a), 11(b) and 11(c), we can see that the cost corresponding to ActGreedy is smaller than that of SGreedy at all the situations. Except in one situation, ActGreedy performs better than LocBased. When the number of input fasta files is 500 and $\omega_t = 0.5$, the cost for the data transfer becomes important. In this case, LocBased performs slightly better than ActGreedy. However, the advantage of LocBased (0.03%) is very small and this may be because of the dynamic changing environment in the Cloud. As the number of input fasta files increases, the advantage of ActGreedy becomes obvious. Compared with LocBased, ActGreedy is up to 4.1% (100 fasta files and $\omega_t = 0.9$), 7.4% (500 fasta files and $\omega_t = 0.9$) and 10.7% (1000 fasta files and $\omega_t = 0.1$) better. Compared with SGreedy, the advantage of ActGreedy can be up to 7.5% (100 fasta files and $\omega_t = 0.5$), 17.2% (500 fasta files and $\omega_t = 0.9$) and 8.8% (1000 fasta files and $\omega_t = 0.5$).

Figures 12 and 13 show the execution and the monetary costs for the execution of SciEvol with different amounts of input data and different values of $\omega_t$. When $\omega_t$ increases, the execution time is largely reduced and the monetary cost increases. When the weight of execution time cost is low, *i.e.* $\omega_t = 0.1$, Compared with LocBased and SGreedy, ActGreedy may correspond to more execution time while it generally takes less monetary cost. When the weight of execution time cost is high, $\omega_t = 0.9$, ActGreedy corresponds to less execution time. The execution with ActGreedy always takes less monetary cost compared with LocBased (up to 14.12%) and SGreedy (up to 17.28%). The reason is that ActGreedy can choose a cheap site to execute fragments, namely the monetary cost to instantiate VMs at that site is low. As a result, ActGreedy makes a good trade-off between execution time and monetary costs for the execution of SciEvol at a multisite cloud.

Furthermore, we measured the amount of data transferred among different sites, which is shown in Figure 14. Since LocBased is optimized for minimizing data transferred between different sites, the amount of intersite transferred data with LocBased remains minimum when the number of input fasta files varies from 100 to 1000. The amount of transferred data corresponding to ActGreedy is slightly bigger than that of LocBased and the difference is between 1.0% and 13.4%. SGreedy has the biggest amount of intersite transferred data. Compared with ActGreedy, the amount of intersite transferred data of SGreedy is up to 122.5%, 139.2% and 148.1% bigger when the number of input fasta files is 10, 500 and 1000. In addition, the amount of data transfer with ActGreedy decreases for the three cases, *i.e.* 500 input fasta files with $\omega_t = 0.9$ and 1000 input fasta files with $\omega_t = 0.5$ or $\omega_t = 0.9$. The reason is that Activities 7 and 8 are scheduled at the same site as Activities 6.5 and 6.6, namely Site 3, which reduces data transfer. Furthermore, when the data transfer rate between different sites decreases, the performance of SGreedy will be much worse since the time to transfer big amounts of data will be much longer.

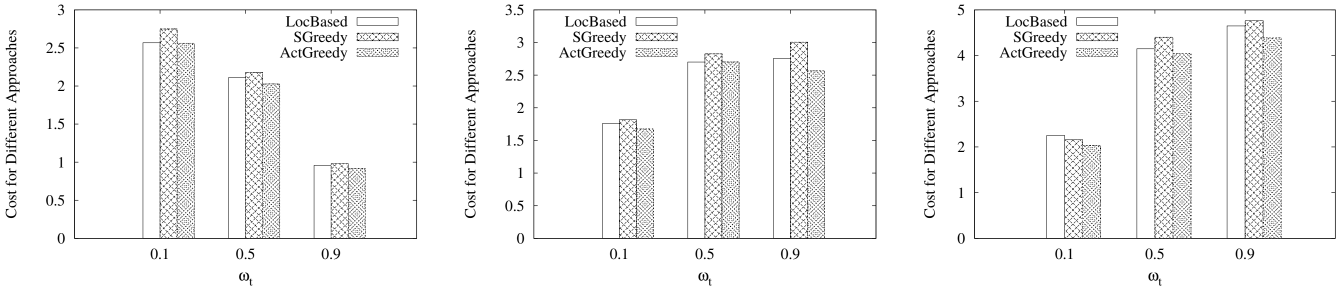
In addition, we measure the idleness of the virtual CPUs according to following formula:

$$Idleness = \frac{\sum_{i=1}^{n} IdleTime(CPU_i)}{\sum_{i=1}^{n} TotalTime(CPU_i)} * 100\% \qquad (7.2.2)$$

where $n$ represents the number of virtual CPUs, $IdleTime$ represents the time when the virtual CPU is not working for the execution of the programs of SciEvol SWf. $TotalTime$ represents the total time that the virtual CPU is instantiated. Figure 15 shows the idleness of virtual CPUs corresponding to different scheduling algorithms and different amounts of fasta files. From the figure, we can see that as $\omega_t$ increases, the idleness becomes bigger. When $\omega_t$ increases, the importance of execution time becomes bigger and more VMs are provisioned to execute fragments. In this case, the time to start the VMs becomes higher compared with execution time. As a result, the corresponding idleness goes up. In addition, when the amount of input files, *i.e.* fasta files, rises, the idleness decreases. The reason is that at this situation, more time is used for the execution of SWf for the increased workload. The figure also shows that LocBased has the smallest idleness while SGreedy has the biggest idleness. This is expected since the VMs at Sites 1 and 2 need to be shut down and restarted during the execution with SGreedy and that the VMs at Site 1 needs to be shut down and restarted during the execution with ActGreedy. The time to restart VMs at a site may consume several minutes while the virtual CPUs are not used for the execution of fragments. Figure 15(a), Figure 15(b) and Figure 15(b) show the experimental results for the corresponding idleness of virtual CPUs. From the figures, we can see that the idleness of ActGreedy is generally bigger than that of LocBased while it is always smaller than that of SGreedy. The idleness of ActGreedy is up to 38.2% ($\omega_t = 0.5$) bigger than that of LocBased and up to 37.8% ($\omega_t = 0.1$) smaller than that of SGreedy for 100 fasta files. The idleness of ActGreedy is up to 12.4% ($\omega_t = 0.1$) bigger than that of LocBased and up to 38.0% ($\omega_t = 0.5$) smaller than that of SGreedy for 500 fasta files. For 1000 fasta files, the idleness of ActGreedy is 18.6% ($\omega_t = 0.1$) and 1.0% ($\omega_t = 0.9$) bigger than that of LocBased. When $\omega_t = 0.5$, the idleness of ActGreedy is 20.9% smaller than that of LocBased. In ad-

[1] The advantage is calculated based on the following formula:

$$Advanage = \frac{Cost_{SGreedy}(\omega_t) - Cost_{ActGreedy}(\omega_t)}{Cost_{ActGreedy}(\omega_t)} * 100\% \qquad (7.2.1)$$

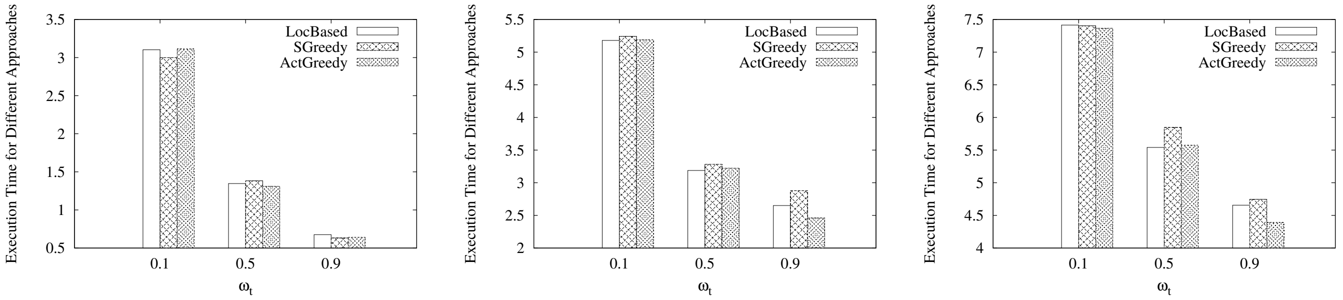


(a) **Comparision for 100 fasta files.**

(b) **Comparision for 500 fasta files.**

(c) **Comparision for 1000 fasta files.**

Figure 11: **Cost for different scheduling algorithms.** According to Formula 5.1.1.

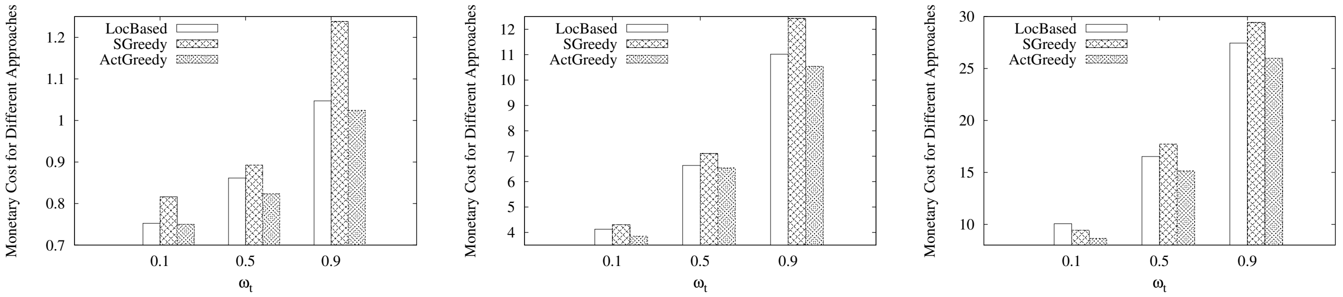


(a) **Execution time (100 fasta files).**

(b) **Execution time (500 fasta files).**

(c) **Execution time (1000 fasta files).**

Figure 12: **Execution time of SciEvol with different scheduling approaches.**

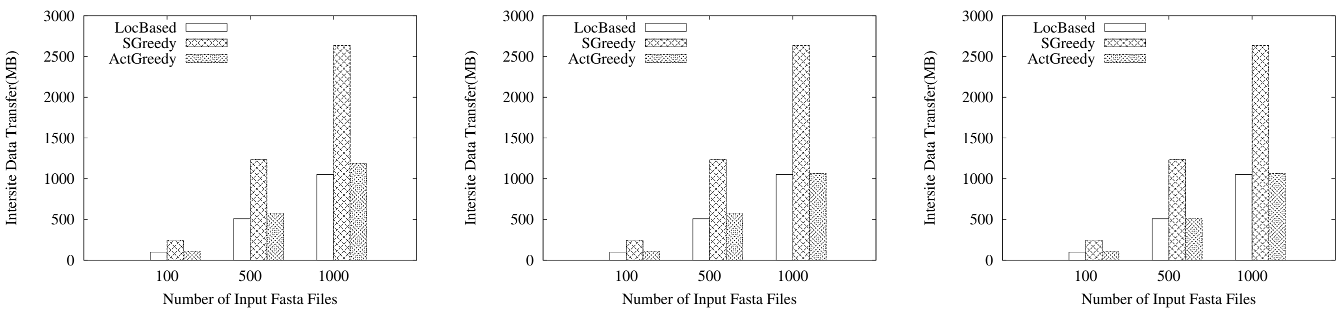


(a) **Monetary cost (100 fasta files).**

(b) **Monetary cost (500 fasta files).**

(c) **Monetary cost (1000 fasta files).**

Figure 13: **Monetary cost of SciEvol execution with different scheduling approaches.**

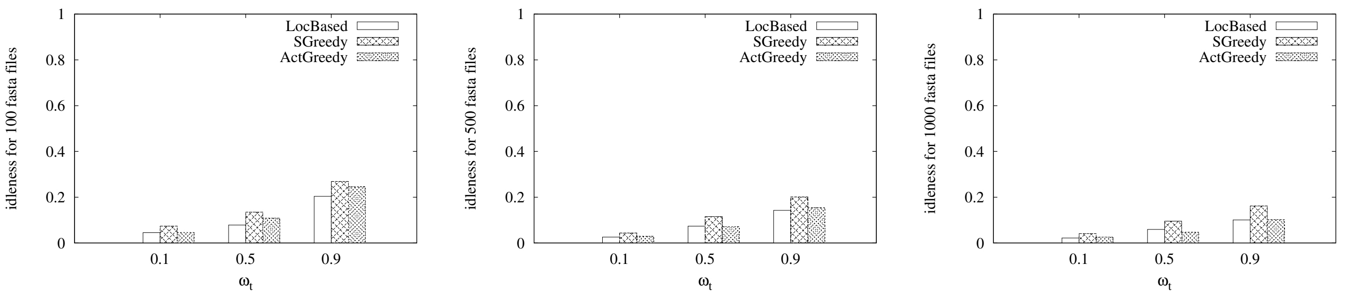


(a) **Intersite data transfer** ($\omega_t$ = 0.1)**.**

(b) **Intersite data transfer** ($\omega_t$ = 0.5)**.**

(c) **Intersite data transfer** ($\omega_t$ = 0.9)**.**

Figure 14: **Intersite data transfer for different scheduling algorithms.**



(a) **Idleness of virtual CPUs for 100 fasta files.** (b) **Idleness of virtual CPUs for 500 fasta files.** (c) **Idleness of virtual CPUs for 1000 fasta files.**

Figure 15: **Idleness of virtual CPUs for different scheduling algorithms.** According to Formula 7.2.2.

Table 12: **Number of generations.**

| Number of sites | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of generations | 1 | 1 | 3 | 10 | 28 | 71 | 162 | 337 | 657 | | |
| Number of activities | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Number of generations | 1 | 1 | 1 | 2 | 6 | 18 | 54 | 162 | 484 | 1450 | 4349 |

dition, the idleness of ActGreedy is up to 50.9% ($\omega_t = 0.5$) smaller than that of SGreedy.

Finally, we study the scheduling time of different algorithms. In order to show the effectiveness of ActGreedy, we compare it with our two other algorithms, *i.e.* SGreedy and LocBased, and two more general algorithms, *i.e.* Genetic and Brute-force.

Table 13: **Comparison of scheduling algorithms.**

| Algorithms | Scheduling time (ms) |
|---|---|
| LocBased | 0.010 |
| SGreedy | 0.014 |
| ActGreedy | 1.260 |
| Genetic | 727 |
| Brute-force | 161 |

An example of scheduling time corresponding to 3 sites and 13 activities is shown in Table 13. This is a small example since the time necessary to schedule the activities may be unfeasible for Brute-force and Genetic when the numbers of activities or sites become high. Then, we vary the numbers of activities or sites. When we increase the number of sites, we fix the number of activities at 13 and when we increase the number of activities, we fix the number of sites to 3. The number of generations for different numbers of activities or different numbers of sites is shown in Table 12. Since the search space gets bigger when the number of activities or sites increases, we increase the number of generations in order to evaluate at least 30% of all the possible scheduling plans for Genetic. We add additional control activities in the SciEvol SWf, which have little workload but increase the search space of scheduling plans. The modified SciEvol SWf is shown in Figure 16 and the scheduling time corresponding to different numbers of activities is shown in 17(a). In addition, we measure the scheduling time corresponding to different numbers of sites while using the original SciEvol SWf, as shown in Figure 17(c). The unit of scheduling time is millisecond. The data constraint remains the same while the number of input files is 100 and $\alpha$ equals to 0.9. In the experiments, only ActGreedy generates the same scheduling plans as that of Brute-force. Since the point of mutation operation and the points of crossover operation of Genetic are randomly selected, the scheduling plans generated by Genetic may not be stable, *i.e.* the scheduling plans may not be the same for each execution of the algorithm. Both LocBased and SGreedy cannot generate the optimal scheduling plans as that of Brute-force.

Table 13 shows that the scheduling time of Genetic and Brute-force is much longer than ActGreedy (up to 577 times and 128 times). Genetic may perform worse than Brute-force for a small number of activities or sites with the specific configuration. The scheduling time of Genetic is smaller than that of Brute-force when the number of activities or sites increases as shown in Figure 17(a) and 17(c). Figure 17(b) shows the scheduling time of different algorithms zooming on 13 − 18 activities, which reveals that Brute-force and Genetic always takes more time to generate scheduling plans than LocBased, SGreedy and ActGreedy. Figures 17(a) and 17(c) show that the scheduling time of ActGreedy, SGreedy and LocBased is much smaller than that of Genetic and Brute-force. The scheduling time of ActGreedy, SGreedy and LocBased is represented by the bottom line in Figures 17(a)17(b)17(c). Even when the number of activities and the number sites is small, the scheduling time is negligible compared with the execution time, it becomes significant when the numbers of activities or sites increase. For instance, with more than 22 activities or 6 sites, the scheduling time of Brute-force exceeds the execution while the scheduling time of ActGreedy remains small. This is because of the high complexity of Brute-force, which is $O(s^{n-f})$. With more than 22 activities or 10 sites, the scheduling time of Genetic is bigger than the execution time. Because of long

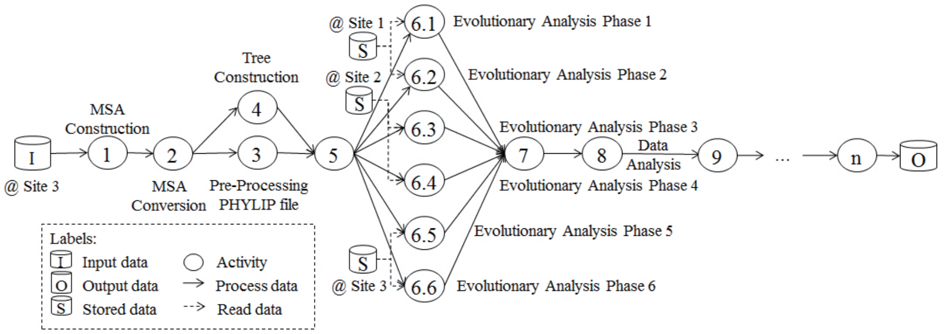
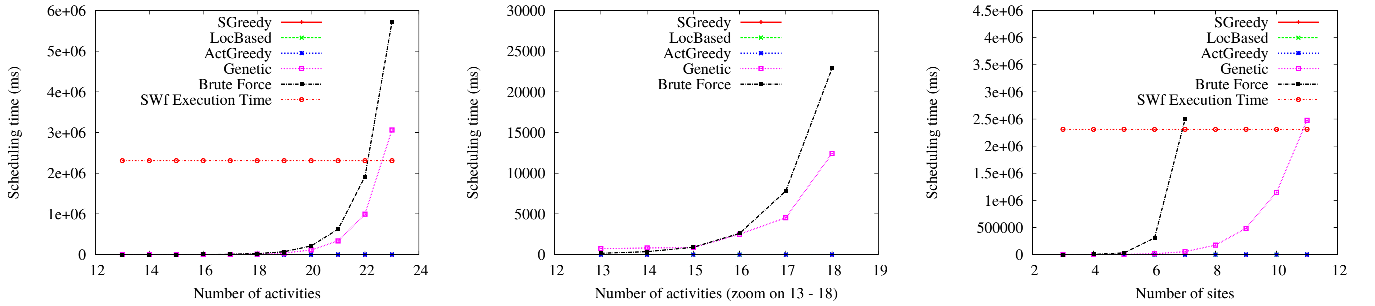


Figure 16: **SciEvol Scientific Workflow.** Activities 9 – *n* are added control activities, which have no workload.



(a) **Scheduling time corresponding to different numbers of activities.**

(b) **Scheduling time (zoom on** 13 – 18 **activities).**

(c) **Scheduling time corresponding to different numbers of sites.**

Figure 17: **Scheduling time.**

scheduling time, Genetic and Brute-force are not suitable for SWfs with a big number of activities, *e.g.* Montage [6] may have 77 activities. In addition, according to [19], 22 activities are below the average number of SWf activities. As a result, Genetic and Brute-force are unfeasible for multisite scheduling of most SWfs. These two algorithms are not suitable for SWf scheduling with a big number of sites. For instance, Azure has 15 sites (regions).

The experimental results show that although ActGreedy may yield more data transferred among different sites and higher idleness (compared with LocBased), it generally yields smaller cost compared with both LocBased and SGreedy and the scheduling time of ActGreedy is much lower than that of Genetic and Brute-force.

## 8. Conclusion

Scientists usually make intensive usage of parallelism techniques in HPC environments. However, it is not simple to schedule and manage executions of SWfs, particularly in multisite cloud environments, which present different characteristics in comparison with single site clouds. To increase the uptake of the cloud model for executing SWfs that demand HPC capabilities provided by multisite clouds and to benefit from data locality, new solutions have to be developed, especially for scheduling SWf fragments in cloud resources. In previous work [14] we have addressed workflow execution in single-site clouds using a scheduling algorithm but these solutions are not suitable for multisite clouds.

In this paper, we proposed a new multi-objective scheduling approach, *i.e.* ActGreedy, for scientific workflows in a multisite cloud (from the same provider). We first proposed a novel multi-objective cost model, based on which, we proposed a dynamic VM provisioning approach, namely SSVP, to generate VM provisioning plans for fragment execution. The cost model aims at minimizing two costs: execution time and monetary costs. Our proposed fragment scheduling approach that is ActGreedy, allows for considering stored data constraints while reducing the cost based on the multi-objective cost model to execute a SWf in a multisite cloud. We used a real SWf that is SciEvol, with real data from the bioinformatics domain as a use case. We evaluated our approaches by executing SciEvol in Microsoft Azure cloud. The results show that since ActGreedy makes a good trade-off between execution time and monetary costs, ActGreedy leads to the least total normalized cost, which is calculated based on the multi-objective cost model, than LocBased (up to 10.7%) and SGreedy (up to 17.2%) approaches. In addition, compared with LocBased (up to 14.12%) and SGreedy (up to 17.28%), ActGreedy always corresponds to less monetary cost since it can choose cheap cloud sites to execute SWf fragments. Furthermore, compared with SGreedy, ActGreedy corresponds to more than two times smaller amounts of transferred data. Additionally, ActGreedy scales very well, *i.e.* it takes a very small time to generate the optimal or near optimal scheduling plans when the number of activities or sites increases, compared with general approaches, *e.g.* Genetic and Brute-force. The results also show that the cost model can estimate the cost within an acceptable error limit and that the provisioning approach (SSVP) generates better provisioning plans for different weights of time cost to execute a fragment at a site, compared with other existing approaches, namely GraspCC. The advantage of SSVP can be up to 115.4%.

## Acknowledgment

Work partially funded by EU H2020 Programme and MCTI/RNP-Brazil (HPC4E grant agreement number 689772), CNPq, FAPERJ, and INRIA (MUSIC project), Microsoft (ZcloudFlow project) and performed in the context of the Computational Biology Institute (www.ibc-montpellier.fr). We would like to thank Kary Ocaña for her help in modeling and executing the SciEvol SWf.

## References

[1] Amazon ec2, amazon elastic compute cloud (amazon ec2). `http://aws.amazon.com/ec2/`.

[2] Microsoft Azure. `http://azure.microsoft.com`, .

[3] Azure cli. `https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-command-line-tools/`, .

[4] Computing capacity for a CPU. `http://en.community.dell.com/techcenter/high-performance-computing/w/wiki/2329`.

[5] Data transfer price. `https://azure.microsoft.com/en-us/pricing/details/data-transfers/`.

[6] Montage. `http://montage.ipac.caltech.edu/docs/gridtools.html`.

[7] Oma genome database. `http://omabrowser.org/All/eukaryotes.cdna.fa.gz`, .

[8] Sequence identifier. `http://omabrowser.org/All/oma-groups.txt.gz`, .

[9] C. Anglano and M. Canonico. Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach. In *22nd IEEE Int. Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8, 2008.

[10] S. Blagodurov, A. Fedorova, E. Vinnik, T. Dwyer, and F. Hermenier. Multi-objective job placement in clusters. In *Proceedings of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis, SC*, pages 66:1–66:12, 2015.

[11] D. Chang, J. H. Son, and M. Kim. Critical path identification in the context of a workflow. *Information & Software Technology*, 44(7):405–417, 2002.

[12] W. Chen, R.F. Da Silva, E. Deelman, and R. Sakellariou. Balanced task clustering in scientific workflows. In *IEEE 9th Int. Conf. on e-Science*, pages 188–195, 2013.

[13] R. Coutinho, L. Drummond, Y. Frota, D. de Oliveira, and K. Ocana. Evaluating grasp-based cloud dimensioning for comparative genomics: A practical approach. In *2014 IEEE Int. Conf. on Cluster Computing (CLUSTER)*, pages 371–379, 2014.

[14] D. de Oliveira, K. A. C. S. Ocaña, F. Baião, and M. Mattoso. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *Journal of Grid Computing*, 10(3):521–552, 2012.

[15] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[16] R. Duan, R. Prodan, and X. Li. Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds. *IEEE Transactions on Cloud Computing*, 2(1):29–42, 2014.

[17] K. Etminani and M. Naghibzadeh. A min-min max-min selective algorihtm for grid task scheduling. In *The Third IEEE/IFIP Int. Conf. in Central Asia on Internet (ICI 2007)*, pages 1–7, 2007.

[18] H. Mohammadi Fard, R. Prodan, and T. Fahringer. Multi-objective list scheduling of workflow applications in distributed computing infrastructures. *Journal of Parallel and Distributed Computing*, 74(3):2152–2165, 2014.

[19] R. Littauer, K. Ram, B. Ludäscher, W. Michener, and R. Koskela. Trends in use of scientific workflows: Insights from a public repository and recommendations for best practice. *International Journal of Digital Curation (IJDC)*, 7(2):92–100, 2012.

[20] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. Parallelization of scientific workflows in the cloud. Research Report RR-8565, 2014.

[21] J. Liu, V. Silva, E. Pacitti, P. Valduriez, and M. Mattoso. Scientific workflow partitioning in multi-site clouds. In *BigDataCloud'2014: 3rd Workshop on Big Data Management in Clouds in conjunction with Euro-Par 2014*, page 12, 2014.

[22] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, pages 1–37, 2015.

[23] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *8th Heterogeneous Computing Workshop*, page 30, 1999.

[24] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26 (6):369–395, 2004.

[25] K.A.C.S. Ocaña, D. de Oliveira, F. Horta, J. Dias, E. Ogasawara, and M. Mattoso. Exploring molecular evolution reconstruction using a parallel cloud based scientific workflow. In *Advances in Bioinformatics and Computational Biology*, volume 7409 of *Lecture Notes in Computer Science*, pages 179–191. 2012.

[26] E. S. Ogasawara, J. Dias, V. Silva, F. S. Chirigati, D. de Oliveira, F. Porto, P. Valduriez, and M. Mattoso. Chiron: a parallel engine for algebraic scientific workflows. *Concurrency and Computation: Practice and Experience*, 25(16):2327–2341, 2013.

[27] D. De Oliveira, K. A. C. S. Ocaña, E. Ogasawara, J. Dias, J. GonçAlves, F. Baião, and M. Mattoso. Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows. *Future Generation Computer Systems*, 29(7):1816–1825, 2013.

[28] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, 2011.

[29] M. Rahman, M. R. Hassan, R. Ranjan, and R. Buyya. Adaptive workflow scheduling for dynamic grid and cloud computing environment. *Concurrency and Computation: Practice and Experience*, 25(13):1816–1842, 2013.

[30] M. A. Rodriguez and R. Buyya. A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds. In *44th Int. Conf. on Parallel Processing, ICPP*, 2015.

[31] I. Sardiña, C. Boeres, and L. de A. Drummond. An efficient weighted bi-objective scheduling algorithm for heterogeneous systems. In *Euro-Par 2009 – Parallel Processing Workshops*, volume 6043, pages 102–111. 2010.

[32] S. Smanchat, M. Indrawan, S. Ling, C. Enticott, and D. Abramson. Scheduling multiple parameter sweep workflow instances on the grid. In *5th IEEE Int. Conf. on e-Science*, pages 300–306, 2009.

[33] X. Sun and Y. Chen. Reevaluating amdahl's law in the multicore era. *Journal of Parallel and Distributed Computing*, 70(2):183–188, 2010.

[34] H. Topcuouglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[35] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the ASKALON grid environment. *SIGMOD Record*, 34 (3):56–62, 2005.

[36] J. Yu, R. Buyya, and C. K. Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *First Int. Conf. on e-Science and Grid Computing*, pages 140–147, 2005.

[37] Z. Yu and W. Shi. An adaptive rescheduling strategy for grid workflow applications. In *IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, 2007.

[38] L. Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE Transactions on Automatic Control*, 8(1):59–60, 1963.