# Scientific Workflow Scheduling with Provenance Data in a Multisite Cloud

Ji Liu, Esther Pacitti, Patrick Valduriez, Marta Mattoso

# Scientific Workflow Scheduling with Provenance Data in a Multisite Cloud

Ji Liu[1], Esther Pacitti[1], Patrick Valduriez[1], and Marta Mattoso[2]

[1] Inria, Microsoft-Inria Joint Centre, LIRMM and University of Montpellier, France
`{ji.liu,patrick.valduriez}@inria.fr`
`{esther.pacitti}@lirmm.fr`
[2] COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
`{marta}@cos.ufrj.br`

**Abstract.** Recently, some Scientific Workflow Management Systems (SWfMSs) with provenance support (*e.g.* Chiron) have been deployed in the cloud. However, they typically use a single cloud site. In this paper, we consider a multisite cloud, where the data and computing resources are distributed at different sites (possibly in different regions). Based on a multisite architecture of SWfMS, *i.e.* multisite Chiron, and its provenance model, we propose a multisite task scheduling algorithm that considers the time to generate provenance data. We performed an extensive experimental evaluation of our algorithm using Microsoft Azure multisite cloud and two real-life scientific workflows (Buzz and Montage). The results show that our scheduling algorithm is up to 49.6% better than baseline algorithms in terms of total execution time.

**Keywords:** Scientific workflow · Scientific workflow management system · Scheduling · Parallel execution · Multisite cloud

## 1 Introduction

Many large-scale *in silico* scientific experiments 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. An SWf is the assembly of scientific data processing activities with data dependencies between them [13]. An activity is a description of a piece of work that forms a logical step within an SWf representation [23] and a task is the representation of an activity within a one-time execution of the activity to process a data chunk. A data chunk is the smallest set of data that can be processed by a program in an activity. Since the tasks of the same activity process different data chunks [23], they are independent. Within one activity, each task processes a data chunk, which has no dependency with other tasks of this activity. Different SWfs, *e.g.* Montage [4] and Buzz [16], can be represented this way since there are many tasks that exploit the same program to process different data chunks. As it takes much time to execute a data-intensive SWf, efficient execution is an important issue.

A Scientific Workflow Management System (SWfMS) is a tool to execute SWfs [23]. Some implementations of SWfMSs are publicly available, *e.g.* Pegasus [15] and Chiron [29]. An SWfMS generally supports provenance data, which is the metadata that captures the derivation history of a dataset, including the original data sources, intermediate datasets, and the workflow computational steps that were applied to produce this dataset [23]. Provenance data, which is used for SWf analysis and SWf reproducibility, may be as important as the scientifc experiment itself [23]. The provenance data is typically stored in a database to provide on-line provenance query [28], and contains the information regarding activities, tasks and files. During the execution of a task, there may be multiple exchanges of provenance data between the computing node and the provenance database.

In order to execute a data-intensive SWf within a reasonable time, SWfMSs generally exploit High Performance Computing (HPC) resources and parallel computing techniques. The HPC resources are generally obtained from a computer cluster, grid or cloud environment. Most of existing SWfMSs have been designed for a computer cluster.

Recently, some SWfMSs with provenance support (*e.g.* Chiron) have been deployed in the cloud. They typically focus on the execution of an SWf at a single cloud site or in even a single computing node [19,20]. Although there are some multisite solutions [17,32], they do not support provenance data, which is important for the analysis of SWf execution. However, the input data necessary to run an SWf may well be distributed at different sites (possibly in different regions), which may not be allowed to be transferred to other sites because of big amounts or proprietary. And it may not be always possible to move all the computing resources (including programs) to a single site for execution. In this paper, we consider a multisite cloud that is composed of several sites (or data centers) of the same cloud provider, each with its own resources and data. In addition, we also take into consideration of the influence of the functionality of provenance data on the SWf multisite execution. The difference between multisite cloud and the environment of single-site or supercomputer is that, in multisite cloud, the data or the computing resources are distributed at different sites and the network bandwidths among different sites are different. In addition, the SWf execution in a multisite cloud is different from the query execution in database and P2P environments because of the programs to execute, security, and diversity.

To enable SWf execution in a multisite cloud with distributed input data, the execution of the tasks of each activity should be scheduled to a corresponding cloud site (or site for short). Then, the scheduling problem is to decide at which sites to execute the tasks in order to achieve a given objective, *e.g.* reducing execution time. Since it may take much time to transfer data between two different sites, the multisite scheduling problem should take into account the resources at different sites, *e.g.* data stored at each site, and different bandwidths between different sties. This is different from the execution at a single site, where data can be shared among different computing nodes and the bandwidths are very big and almost the same for different nodes. In addition, the time to transfer the provenance data cannot be ignored in a multisite cloud environment, which is different from the execution without provenance data. Compared with the approach of scheduling activities in a multisite environment [25], the task scheduling is fine-grained, which enables the execution of the same activity at different sites to deal with distributed data and programs.

We focus on the task scheduling problem to reduce the makespan, *i.e.* the execution time, of executing an SWf in a multisite cloud. We use a distributed SWfMS architecture with a master site that coordinates the execution of each site and that stores all the provenance data of the SWf execution. In this architecture, the intersite transferred data can be intermediate data or provenance data produced by the SWf execution. The intermediate data is the data generated by executing activities and can also be the input data for the tasks of following activities. In the multisite cloud, the bandwidth between two different sites (of different regions) may be small. For data-intensive SWfs, there may be many data, *e.g.* intermediate data and provenance data, to transfer across different sites for the execution of a task while the time to execute the task can be very small, *e.g.* a few seconds or even less than one second. As a result, the time to transfer intermediate data and the time to generate the provenance data cannot be ignored in the scheduling process. Thus, we also consider the time to transfer both the intermediate data and the provenance data in the scheduling process in order to better reduce the overall execution time of SWf execution.

The difference between our work and others is multisite execution with provenance support. In the paper, we make the following contributions. First, we propose multisite Chiron, with a novel architecture to execute SWfs in a multisite cloud environment while generating provenance data. Second, an extended multisite provenance model and global provenance management of distributed provenance data in a multisite cloud. Third, we propose a novel multisite task scheduling algorithm, *i.e.* Data-Intensive Multisite task scheduling (DIM), for SWf execution with provenance support in multisite Chiron. Fourth, we carry out an extensive experimental evaluation, based on the implementation of multisite Chiron in Microsoft Azure, and using two real SWf use cases (Buzz and Montage). This paper is a major extension of [24], with more details on related work and problem definition and the adaptation of single site Chiron to multisite Chiron. The added value compared with [24] is about 40% and the main differences are in: the model to estimate the time to execute a bag of tasks (Section 5.2); the experiments of Montage and the improvement of the implementation of scheduling algorithms (Section 6.3); the complexity analysis (Section 5.3) and the convergence analysis of DIM algorithm (Appendix).

This paper is organized as follows. Section 2 introduces the related work. Section 3 presents the problems for task scheduling of SWf execution in a multisite cloud environment. Section 4 gives the design of a multisite SWfMS. Section

5 explains our proposed scheduling algorithm. Section 6 gives our experimental evaluation. Section 7 concludes the paper.

## 2 Related Work

Classic scheduling algorithms, *e.g.* Opportunistic Load Balancing (OLB) [26], Minimum Completion Time (MCT) [26], min-min [18], max-min [18] and Heterogeneous Earliest Finish Time (HEFT) [38], address the scheduling problem for the objective of reducing execution time within a single site. The OLB algorithm randomly assigns each task to an available computing node without considering the feature of the task or the computing node. The MCT algorithm schedules each task to the computing node that can finish the execution first. HEFT gives the priority to each task according to the dependencies of tasks and the workload of the task. Then, it schedules the tasks with the highest priority to the computing node that can finish the execution first. The min-min algorithm schedules the task, which takes the least time to execute, to the computing node that can finish the execution first. The max-min algorithm schedules the task, which takes the biggest time to execute, to the computing node that can finish the execution first. Since the size of each data chunk of the same activity is similar and that the tasks of the same activity exploit the same program, it is reasonable to assume that the tasks of the same activity have the same workload. We also assume that the tasks of the same activity are independent since each task processes a data chunk. Thus, the HEFT, min-min and max-min algorithms degrade to the MCT algorithm for this kind of tasks. Some other solutions [35,37,39] for SWf scheduling also focus on single site execution. These techniques do not consider the time to generate provenance data. Dean and Ghemawat [12] propose to schedule tasks to where the data is. Although this method focuses on single site, it considers the cost to transfer data among different computing nodes. However, this algorithm depends on the location of data. When the data is not evenly distributed at each computing node, this algorithm may lead to unbalanced load at some computing nodes and long execution time of tasks. De Oliveira *et al.* [11] propose a provenance based task scheduling algorithm for single site cloud environments. Some adaptation of SWfMSs [6,9] in the cloud environment can provide the parallelism in workflow level or activity level, which is coarse-grained, at a single site cloud. These methods cannot perform parallelism of the tasks of the same activities and they cannot handle the distributed input data at different sites.

A few multisite scheduling approaches are proposed, but they do not consider the distribution of input data at different sites and have no support for provenance data, which may incur much time for intersite data transfer. Duan *et al.* [17] propose a multisite multi-objective scheduling algorithm with consideration of different bandwidths in a multisite environment. However, they do not consider the input data distribution at different sites and do not provide provenance support, which may incur much time for intersite provenance data transfer. In previous work [21,25], we proposed solutions of multisite activity scheduling of SWfs. However, the activity scheduling method is coarse-grained: it can schedule the entire execution of each activity to a site but cannot schedule different tasks of one activity to different sites. Thus, it cannot handle the distributed input data of an SWf in a multisite cloud environment. Luis *et al.* [32] propose caching metadata in the memory and replicating the metadata for SWf execution in a multisite cloud. The metadata is the description information of files at each site. In this method, data transfer is analyzed in multi-site SWf execution, stressing the importance of optimizing data provisioning. However, the metadata is not yet explored on task scheduling and, they just simulated the SWf execution in the experiments. Their method can be used to optimize the metadata management in our multisite Chiron in the future.

There is also some work in scheduling tasks for query optimization with the consideration of data distribution in databases [7,8,30]. However, the major difference between SWf scheduling and query optimization lies in the kind of programs used to process data. The programs used in a query are user-defined functions and typically run within the database system while the programs in SWfs are typically black box code managing their own data outside the scope of the SWfMS. Furthermore, the query execution generates data within the database while the programs in SWfs generate files, which are processed by I/O operations.

Compared with P2P [27,31], a major difference is that multisite cloud does not have as many sites. Another difference is that the security issue in multisite cloud is more important than in P2P, *e.g.* some data cannot be moved to another site.

## 2.1 Single Site Chiron

Chiron [29] is an SWfMS for the execution of data-intensive SWfs at a single site, with provenance support. At a single site, Chiron takes one computing node as a master node and the other nodes as slave nodes, as shown in Figure 1. In a cloud environment, a computing node is a Virtual Machine (VM). Designed for HPC environments, Chiron relies on a Shared File System [1], *e.g.* Network File System (NFS) [33], for managing data. All the computing nodes in the cluster can read or write the data stored in the shared file system. Chiron exploits a relational database, *i.e.* PosgreSQL, to store provenance data.

There are six modules, *i.e.* textual UI, activity manager, single site task scheduler, task executor, provenance data manager and shared file system, in the single site Chiron. The users can use a textual User Interface (UI) to interact with Chiron, in order to start an instance of Chiron at each computing node. During the execution of an SWf, each activity and its dependencies are analyzed by the activity manager to find executable activities, *i.e.* unexecuted activities, of which the input data is ready [29]. In order to execute an activity, the corresponding tasks are generated by the activity manager. Afterwards, the task

[1] In a shared file system, all the computing nodes of the cluster share some data storage that are generally remotely located [22].

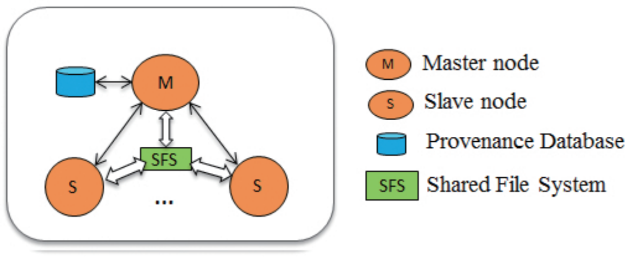

Fig. 1: **Architecture of single site Chiron.**

scheduler schedules each task to a computing node. During SWf execution, the tasks of each activity are generated independently and the scheduling of the tasks of each activity is done independently. The scheduling process is performed at the beginning of the execution of each activity when the tasks are generated and to be scheduled at each site. At the beginning of each activity, the corresponding tasks of this activity are generated and scheduled without interaction with other activities. In single site Chiron, each time a slave node is available, it requests new tasks from the master node, which in turn searches for unexecuted tasks and dispatches them to the slave. This approach is efficient for single site implementations, where communication latency is negligible and there is a shared file system. Then, the task execution module at each computing node executes the corresponding scheduled tasks. When all the tasks of the executable activity are executed, the activity manager analyzes the activities to find new executable activities to execute. The process of activity analysis, task scheduling and task execution are repeated until all activities have been executed. Since the input data, intermediate data and output data of SWfs are stored in a shared file system, Chiron does not need to manage data transfer between different computing nodes. During SWf execution, the activity manager, the task scheduler and the task executor generate provenance data, which is gathered by the provenance data manager. The provenance data manager is located at the master node of the cluster.

The single site provenance model [29] is shown in Figure 2. In this model, an SWf is composed of several activities. An activity has an operator, *i.e.* the program for this activity. The status of the activity can be ready, running or finished. The *activationCommand* of an activity is to execute the activity. The *extractorCommand* is to generate provenance data for the corresponding tasks. The time at which the activity execution starts is *executionStart* and the time at which it ends is *executionEnd*. One activity is related to several tasks, input relations and output relations. One relation is the input or output parameters for the activity. Each relation has its own attributes and tuples. The tasks of an activity are generated based on the input relation of the activity. A task processes the files associated with the corresponding activity. Each task has a status, *i.e.* ready, running or finished. In addition, the start time and end time of its execution is recorded as *ExecutionStart* and *ExecutionEnd*. During

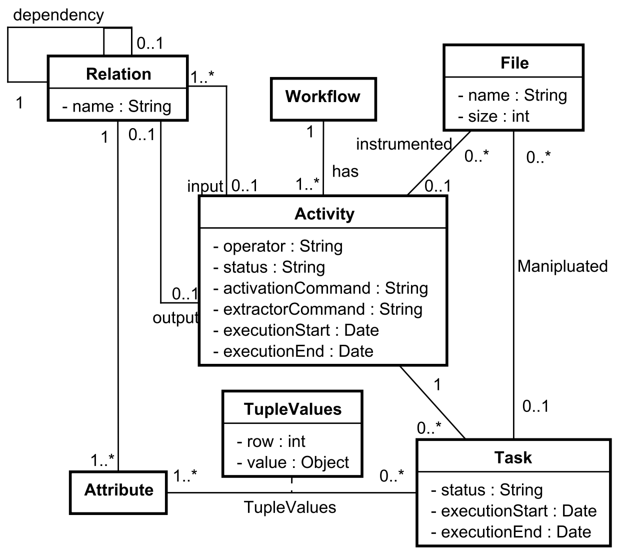

Fig. 2: **Single Site Provenance Model [29].**

execution, the corresponding information of activities, files and tasks are stored as provenance data.

## 3 Problem Definition

This section introduces some important terms, *i.e.* SWf and multisite cloud, and formally defines the task scheduling problem we address.

An SWf is the assembly of activities and data dependencies where the activities are connected by data dependencies, *i.e* there is at least a path between every pair of activities. The path is a combination of data dependencies without the consideration of the direction between two activities. An SWf is generally represented as a Directed Acyclic Graph (DAG). Let us denote an SWf by $W(V,E)$. $V = \{v_1, v_2, ..., v_n\}$ represents a set of $n$ vertices, which represent the scientific data processing activities. $E = \{e_{i,j}$: $v_i, v_j \in V$ and Activity $v_j$ consumes the output data of Activity $v_i$ $\}$ represents a set of edges that correspond to dependencies between activities in $V$. Activity $v_i$ is the parent activity of Activity $v_j$ and Activity $v_j$ is the child activity of Activity $v_i$. If it has no parent activity, an activity is a start activity. If it has no child activity, an activity is an end activity. If it has both parent activity or child activity, an activity is an intermediate activity. Since an activity may process big amount of data, it corresponds to multiple tasks. Thus, as shown in Figure 3, Activity $A_k$ may have $n$ tasks $\{t_1, t_2, ..., t_n\}$, each consuming a data chunk produced by the tasks of parent activities of Activity $A_k$, *i.e.* Activities $A_i$ and $A_j$. For data-intensive SWfs, the time to transfer data cannot be ignored compared with the time to process data.

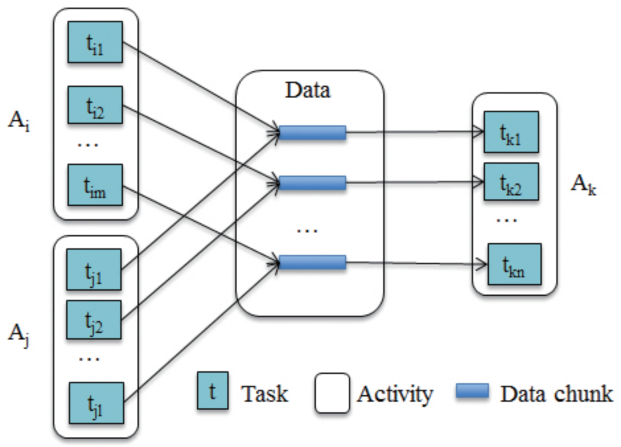

Fig. 3: **Activity and tasks.**

A multisite cloud $MS(S) = \{S, Conf\}$ is a cloud with multiple sites (data centers) of the same cloud provider, each being explicitly accessible to cloud users. $S = \{s_1, s_2, ..., s_m\}$ represents a set of sites while $Conf = \{c_1, c_2, ..., c_m\}$ represents the cloud configurations of each cloud site for the user. A cloud configuration $c_i$ is a combination of parameters of diverse available resources allocated to the user at a site, which may contain information about quality, quantity, and types of the resources. A multisite cloud configuration defines the instances of VMs and storage resources for the users at a multisite cloud. In this paper, a cloud site corresponds to a combination of resources, *e.g.* a cluster of VMs, data and cloud services, which are physically located at a data center. The cloud services can be database or message queue service. In the cluster of VMs, each VM is a computing node.

We assume that the input data of an SWf is distributed at different sites and cannot be moved across different sites because of the proprietary and the security of the data stored at the site. In addition, some laws also restrict the movement of data among different regions. Thus, the tasks of the start activity should be scheduled at the site where the data is. Moreover, we assume that the intermediate data can be moved across different sites and that the time to transfer the input data of tasks between two different sites and the time to generate provenance data is non-negligible compared with the execution time of a task. During the execution, the input data of each activity can be distributed at different sites. Thus, the tasks of the intermediate activities or end activities can be scheduled at any site. In this paper, we focus on the task scheduling of intermediate and end activities.

Let $T$ be a bag of tasks corresponding to the same activity. Then, the scheduling process of $T$ is to choose the sites in $S$ to execute the tasks in $T$, *i.e.* mapping each task to an execution site, while the input data of each task is distributed at different sites. A scheduling plan ($SP(T, S)$) is a mapping between tasks and sites, which specifies which task is executed at which site. Based on an estimation of execution time (see details in Section 5.2), the problem we address is the following [30]:

min(TotalTime($SP(T, S)$))

subject to

distributed input data

The decision variable is $SP(T, S)$, which is defined as a hash table (key-value pairs): $SP(T, S) = \{(t_1, s_2), (t_2, s_1), ..., (t_n, s_m)\}$ where $t_i \in T$ and $s_j \in S$.

Thus, the task scheduling problem is how to generate a scheduling plan $SP(T, S)$ that reduces TotalTime($SP(T, S)$) of all the tasks in $T$ with consideration of distributed input data and provenance data. The distribution of the input data of each task can be also represented as a hash table, *e.g.* Dist($t$, $S$) $= \{(F_1, s_2), (F_2, s_3), ..., (F_k, s_m)\}$ ($F_i$ is the name of a file and $s_j \in S$), which is known before execution. Dist($t$, $S$) represents the distribution of the input file of Task $t$ in the multisite $S$. The key is the name of the corresponding file and the value is the name of the site where the data of the file is stored.

## 4 System Design

In this section, we present the distributed architecture of multisite Chiron, with the modifications to adapt the single site Chiron to a multisite Cloud. Multisite Chiron can manage the communication of Chiron instances at each site and automatically take advantage of resources distributed at each site to process the distributed input data.

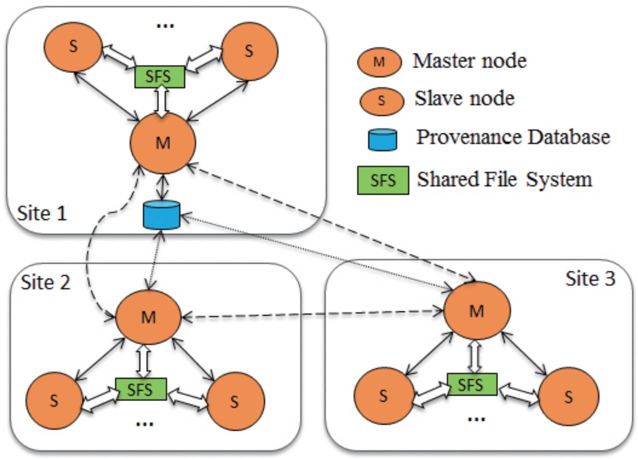

Fig. 4: **Architecture of multisite Chiron.**

In the execution environment of multisite Chiron, there is a master site (site 1 in Figure 4) and several slave sites (Sites 2 and 3 in Figure 4). The master site is similar to the execution environment of a single site Chiron with computing nodes, shared file system and a provenance database. Moreover, a queuing service (see Section 6.2) is deployed at the master site. A slave site is composed of a cluster of VMs with a deployed shared file system. In addition, the master node of each site is configured to enable the message communication and data transfer with other sites. Furthermore, the master node also schedules the tasks at each site and transfers the provenance data generated by each node to a centralized database.

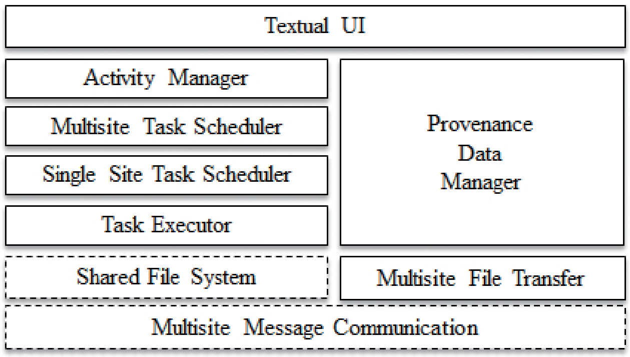

Fig. 5: **Multisite Layered Architecture.**

The layered architecture of multisite Chiron is depicted in Figure 5. The textual UI is present at each node of each site to start an instance of Chiron. The activity manager is located at the master node of the master site to analyze the activities to find executable activities. The multisite task scheduler is located at the master node of the master site, which schedules the tasks to be executed. The provenance data manager works at the master node of each site to gather provenance data for the tasks executed at each site and updates the provenance data in the provenance database. The task executor is present at each node of each site to execute tasks. The shared file system is deployed at the master node of each site and is accessible to all the nodes of the same site. The multisite file transfer and multisite message communication work at the master node of each site to enable the communication of different sites. The other modules are the same as presented in Section 2.1.

In a multisite cloud, multisite Chiron analyzes the data dependencies of each activity. When the input data of an activity is ready [29], it generates tasks. Then, the tasks of each activity are independently scheduled to each site. All the previous processes are realized at the master node of the master site. Then, the data is transferred to the scheduled sites and the tasks are executed at the scheduled sites. Although the input data of an SWf cannot be moved, the

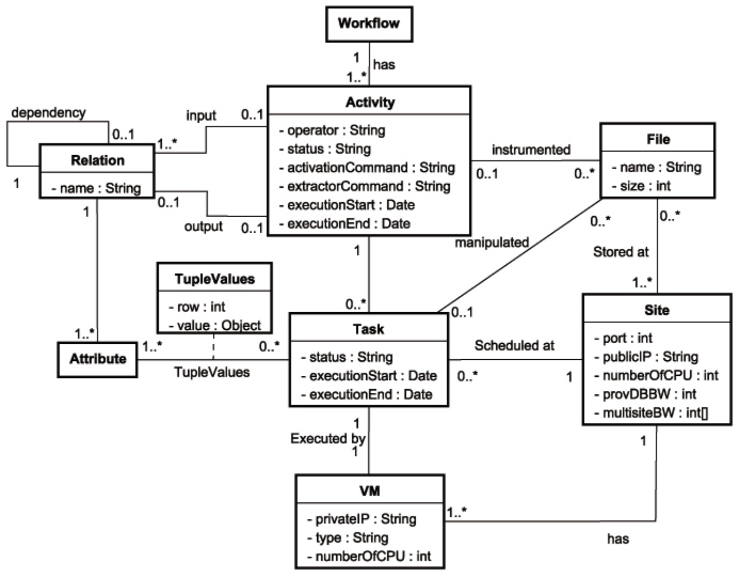

Fig. 6: **Multisite Provenance Model.**

intermediate data can be moved. After the execution of tasks, the provenance data [29] of each site are transferred to the provenance database. When the tasks of all the activities are executed, the execution of an SWf is finished.

In order to extend Chiron to a multisite environment, three key aspects, i.e. provenance model, multisite communication and multisite scheduling, must be considered. First, we adapt the provenance model to the multisite environment. As shown in Figure 6, we add the information about site and computing node (VM) into the provenance model. A site has its own public IP address, public ports for the communication with other sites, number of virtual CPUs, bandwidth to transfer data to the provenance database and bandwidth to transfer data to other sites. A site can contain several VMs. Each VM has its private IP address (which can only be recognized by the devices deployed in the same Web domain), the type of VM, and the number of virtual CPUs. The type of a VM is configured by a cloud user. In a multisite environment, the provenance database is located at a master site while the provenance data is directly stored in the local file system of the master node, which is not the shared file system, because of good performance. Since one task is executed at one computing node of a specific site, a task is related to one computing node and one site. A file can be stored at several sites. Since the input data of a task may be stored at one site (Site $s_1$) and processed at another site (Site $s_2$), it is transferred from $s_1$ to $s_2$ before being processed. As a result, the data ends up being stored at the two sites. Thus, one file is related to several sites. In addition, the provenance data can provide data location information for the scheduling process. As a result, users can also get execution information, *i.e.* which task is executed at which site, from the provenance database. The other objects and relationships remain the same as in the single site provenance model as presented in Section 2.1.

In multisite environments, the provenance data is stored at each site first and then transferred to the centralized provenance database asynchronously with the execution of other tasks. In long running SWfs, provenance data needs to be queried at runtime for monitoring. Since it is convenient for the users to analyze the provenance data in a centralized database, we choose a centralized site, *i.e.* the master site, to store provenance data. In a multisite environment, the provenance data transfer may have a major inuence on the data transfer of task scheduling. Latency hiding techniques can be used to hide the time to transfer data but it is difficult to hide the time to transfer the real-time provenance data generated during execution. Overall, the multisite scheduling problem should take into account the resources at different sites and intersite data transfer, including intermediate data to be processed by tasks and the provenance data.

Second, to support communication between different sites, we add two modules, *i.e.* multisite message communication module and multisite file transfer module. The multisite message communication module is responsible for the exchange of control messages among different sites. The control messages are generated for synchronizing the execution of each site and sharing information among different sites. The synchronization ensures that the activities are executed after their input is ready, *i.e.* after their parents activities are executed, at each site. The multisite file transfer module transfers files to be processed by a task from the site where the files are stored to the site where the task is executed. In fact, the architecture of Chiron is the combination of master-worker model and peer-to-peer model. The master-worker model is responsible for the synchronization among different sites by message communication while the peer-to-peer model is used to share data among sites through multisite file transfer module. The implementation techniques of the two modules are detailed in Section 6.2.

Third, we provide a multisite task scheduling module in multisite Chiron, which is detailed in Section 5.

## 5 Task Scheduling

In this section, we present propose a multisite task scheduling algorithm, *i.e.* Data-Intensive Multisite task scheduling (DIM). Then, we present the method to estimate the total time of a bag of tasks at a single site cloud, which is used in the DIM algorithm. Finally, we analyze the complexity of the DIM algorithm.

### 5.1 Multisite Task Scheduling

Multisite task scheduling is done with a two Level (2L) approach (see Figure 7) because of small complexity. The first level performs multisite scheduling, where each task is scheduled to a site. In this paper, we focus on this level and propose DIM, which is designed for this level. Then, the second level performs single site scheduling, where each task is scheduled to a computing node of the site

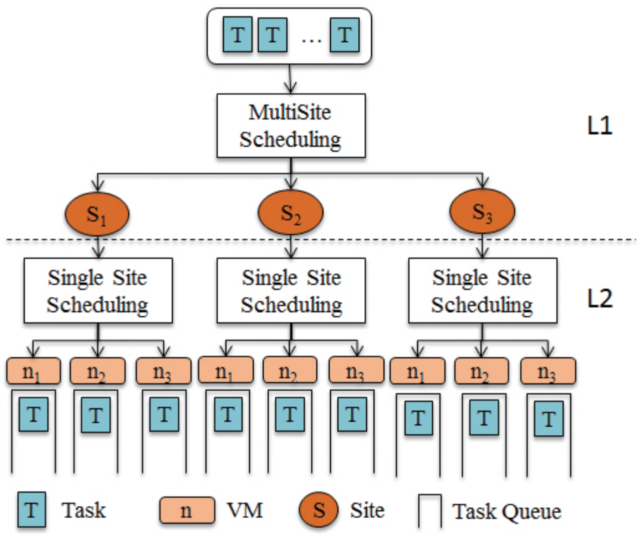

Fig. 7: **Multisite Scheduling.** The master node at the master site schedules tasks to each site. At each site, the master node schedules tasks to slave nodes.

by the default scheduling strategy (dynamic FAF [29]) of Chiron. A task processes an input data chunk within an activity. Different tasks process different data chunks. When an activity has $n$ input data chunks, $n$ tasks are executed independently. Synchronization is based on data dependency between activities as defined at the SWf specification. In the experiments we forced a synchronization so that the next activity only starts after all tasks of the previous activity are executed. Compared with a one Level (1L) approach that schedules tasks directly to computing nodes at different cloud sites, this 2L approach may well reduce the multisite scheduling complexity (see Section 5.3). According to [34], 2L scheduling also significantly outperforms 1L scheduling in terms of performance, *i.e.* the speedup of execution. In addition, the 2L scheduling approach can exploit the existing scheduling solutions of single site SWfMSs. In this paper, we focus on the multisite scheduling part, since we use the default scheduling solutions of Chiron for single site scheduling. However, even in the 2L scheduling approach, the solution space of the multisite level scheduling is still very large and complex.

In our layered architecture (see Section 4), the multisite scheduling is performed at the master node of the master site. For the tasks of data-intensive SWfs, the time to transfer task input data and the time to generate provenance data should not be ignored, in particular in case of low bandwidth of intersite connection and big amounts of data in the files to be transferred between different sites. This is why we consider the time to transfer task input data and provenance data in the scheduling process. The method to estimate the total time of a bag of tasks at a single site is detailed in Section 5.2. In addition,

**Algorithm 1** Data-Intensive Multisite task scheduling (DIM)

```
Input: $T$: a bag of tasks to be scheduled; $S$: a set of cloud sites
Output: $SP(T, S)$: the scheduling plan for $T$ in $S$
 1: $SP(T, S) \leftarrow \emptyset$
 2: for each $t \in T$ do
 3:     $s \leftarrow GetDataSite(\mathrm{Dist}(t, S))$
 4:     $SP(T, S) \leftarrow SP(T, S) \cup \{(t, s)\}$
 5:     TotalTime( $SP(T, S)$ )
 6: while MaxunbalanceTime( $SP(T, S)$ ) is reduced in the last loop do
 7:     $sMin \leftarrow MinTime(S)$
 8:     $sMax \leftarrow MaxTime(S)$
 9:     $SP(T, S) \leftarrow$ TaskReschedule($sMin$, $sMax$, $SP(T, S)$)
end
```

during the scheduling, if the data cannot be moved, the associated tasks are scheduled at the site where the data is stored. As explained in Section 2.1, the tasks of each activity are generated and scheduled independently.

The DIM algorithm schedules a bag of tasks onto multiple sites (see Algorithm 1). Since it takes much time to transfer data among different sites, we first schedule the tasks to where their input data is stored in DIM. However, after this first step, when the data is evenly distributed, workload at each site may be unbalanced, which leads to bigger execution. Thus, at the second step, we adjust the scheduling of tasks until load balance is achieved among different sites so as to reduce the execution time. The details are explained as follow. First, the tasks are scheduled according to the location of input data, *i.e.* the site that stores the biggest amount of input data (Lines 2-5), which is similar to the scheduling algorithm of MapReduce [12]. Line 3 searches the site that stores the biggest part of input data corresponding to Task $t$. Line 4 schedules Task $t$ at Site $s$. The scheduling order (the same for Algorithm 2) is based on the *id* of each task. Line 5 estimates the total time of all the tasks scheduled at Site $s$ with consideration of generating provenance data and intersite data transfer according to Formula 3. Then, the total time at each site is balanced by adjusting the whole bag of tasks scheduled at that site (lines 6-9). Line 6 checks if the maximum difference of the estimated total time of tasks at each site can be reduced by verifying if the difference is reduced in the previous loop or if this is the first loop. While the maximum difference of total time can be reduced, the tasks of the two sites are rescheduled as described in Lines 7-9. Lines 7 and 8 choose the site that has the minimal total time and the site that has the maximum total time, respectively. Then, the scheduler calls the function $TaskReschedule$ to reschedule the tasks scheduled at the two selected sites to reduce the maximum difference of total time.

In order to achieve load balancing of two sites, we propose $TaskReschedule$ algorithm. Let us assume that there are two sites, *i.e.* Sites $s_i$ and $s_j$. For the tasks scheduled at each site, we assume that the total time of Site $s_i$ is bigger than Site $s_j$. In order to balance the total time at Sites $s_i$ and $s_j$, some of the

**Algorithm 2** Tasks Rescheduling

```
Input: $s_i$: a site that has bigger total time for its scheduled tasks; $s_j$: a site that has
    smaller total time for its scheduled tasks; $SP(T,S)$: original scheduling plan for a
    bag of tasks $T$ in multisite $S$
Output: $SP(T,S)$: modified scheduling plan
1: $Diff \leftarrow CalculateExecTimeDiff(s_i, s_j, SP(T,S))$        ▷ Absolute value
2: $T_i \leftarrow GetScheduledTasks(s_i, SP(T,S))$
3: for each $t \in T_i$ do
4:     $SP'(T,S) \leftarrow ModifySchedule(SP(T,S), \{(t, s_j)\}$
5:     $Diff' \leftarrow CalculateExecTimeDiff(s_i, s_j, SP'(T,S))$        ▷ Absolute value
6:     if $Diff' < Diff$ then
7:         $SP(T,S) \leftarrow SP'(T,S)$
8:         $Diff \leftarrow Diff'$
end
```

tasks scheduled at Site $s_i$ should be rescheduled at Site $s_j$. Algorithm 2 gives the method to reschedule a bag of tasks from Site $s_i$ to Site $s_j$ in order to balance the load between the two sites. Line 1 calculates the difference of the total time of two sites according to Formula 3 with a scheduling plan. In the function $CalculateExecTimeDiff(s_i, s_j, SP(T,S))$, based on the scheduling plan ($SP(T,S)$) and the method to estimate the total execution of tasks at a single site (see Section 5.2), the total execution time of Site $s_i$ and $s_j$ can be calculated. Then, the difference ($Diff$) of the total execution between the two sites can also be calculated. Line 2 gets all the tasks scheduled at Site $s_i$. For each Task $t$ in $T_i$ (line 3), it is rescheduled at Site $s_j$ if the difference of total time of the two sites can be reduced (lines 4-8). The task that has no input data at Site $s_j$ is rescheduled first. Line 4 reschedules Task $t$ at Site $s_j$. Line 5 calculates the total time at Sites $s_i$ and $s_j$. Lines 6-7 updates the scheduling plan if it can reduce the difference of total time of the two sites by rescheduling Task $t$ and if the total time of Site $s_i$ is still bigger than or equal to that of Site $s_j$.

## 5.2 Estimation of Execution Time

In this section, we propose the model to estimate the time to execute a bag of tasks in multiple sites and the time to execute the scheduled tasks at each site. The time to execute a bag of tasks in multiple sites can be expressed as the maximum time to execute all the tasks scheduled among each site as shown in Formula 1.

$$TotalTime(SP(T,S)) = \max_{T_i \subset T, s_i \subset S, 0 \le i \le m} TotalTime(T_i, s_i) \tag{1}$$

$T_i$ represents the bag of tasks scheduled at Site $s_i$ according to the scheduling plan $SP(T,S)$ and $m$ represents the number of sites.

We now give the method to estimate the total time to execute a bag of tasks at a single site, which is used in both the DIM algorithm and the MCT algorithm to achieve load balancing of different sites. Formula 2 gives the estimation of total time without considering the time to generate provenance data, which is used in the MCT algorithm.

$$\begin{aligned} TotalTime(T, s) =& ExecTime(T, s) \\ &+ InputTransTime(T, s) \end{aligned} \tag{2}$$

$T$ represents the bag of tasks scheduled at Site $s$. $ExecTime$ is the time to execute the bag of Tasks $T$ at Site $s$, *i.e.* the time to run the corresponding programs. $InputTransTime$ is the time to transfer the input data of the tasks from other sites to Site $s$. In the DIM algorithm, we use Formula 3 to estimate the total time with consideration of the time to generate provenance data.

$$\begin{aligned} TotalTime(T, s) =& ExecTime(T, s) \\ &+ InputTransTime(T, s) \\ &+ ProvTransTime(T, s) \end{aligned} \tag{3}$$

$ProvTransTime$ is the time to generate provenance data in the provenance database.

We assume that the workload of each task of the same activity is similar. The average workload (in GFLOP) of the tasks of each activity and the computing capacity of each VM at Site $s$ is known to the system. The computing capacity (in GFLOPS) indicates the workload that can be realized per second. Then, the time to execute the tasks can be estimated by dividing the total workload by the total computing capacity of site $s$, as shown in Formula 4.

$$ExecTime(T, s) = \frac{|T| * AvgWorkload(T)}{\sum_{VM_i \in s} ComputingCapacity(VM_i)} \tag{4}$$

$|T|$ represents the number of tasks in Bag $T$. $AvgWorkload$ is the average workload of the bag of tasks.

The time to transfer input data can be estimated as the sum of the time to transfer the input data from other sites to Site $s$ of each task as in Formula 5.

$$InTransTime(T, s) = \sum_{t_i \in T} \sum_{s_i \in S} \frac{InDataSize(t_i, s_i)}{DataRate(s_i, s)} \tag{5}$$

$InDataSize(t_i, s_i)$ represents the size of input data of Task $t_i$, which is stored at Site $s_i$. The size can be measured at runtime. $DataRate(s_i, s)$ represents the data transfer rate, which can be configured by users. $S$ represents the set of sites.

Finally, the time to generate provenance data is estimated by Formula 6.

$$\begin{aligned} ProvTransTime(T, s) =& |T| * TransactionTimes(T) \\ &* AvgTransactionTime(s) \end{aligned} \tag{6}$$

$|T|$ represents the number of tasks in Bag $T$. We can estimate $AvgTransactionTime$ by counting the time to perform a transaction to update the provenance data of a task at the provenance database from Site $s$. $TransactionTimes(T)$ represents the number of transactions to perform for generating the provenance data of each task in Bag $T$. It can be configured according to the features of the SWfMS.

## 5.3 Method Analysis

In this section, we analyze the complexity of the 2L scheduling and the DIM algorithm. Let us assume that $N$ ($N >> 2$) tasks are scheduled to $M$ ($M > 2$) sites, each of which has $K$ ($K > 2$) computing nodes. The complexity of the 2L approach is $M^N + K^N$, where $M^N$ is the complexity of the multisite level and $K^N$ is the complexity of the single site level. Assume that there are $N_i$ tasks scheduled at site $s_i$ while $\sum_{i=1}^{M} N_i = N$. Thus, the complexity of single site scheduling is:

$$\begin{aligned}\prod_{i=1}^{M} K^{N_i} &= K^{\sum_{i=1}^{M} N_i} \\ &= K^N\end{aligned} \tag{7}$$

Thus, the complexity of the single site scheduling of the 2L approach is $K^N$. However, the complexity of the 1L approach is $(M * K)^N$. Let us assume that $N > 2, M > 2$ and $K > 2$.

$$\begin{aligned}M^N + K^N &< (\frac{1}{2} * M * K)^N + (\frac{1}{2} * M * K)^N \\ &< (\frac{1}{2})^{(N-1)} * (M * K)^N \\ &< (M * K)^N\end{aligned} \tag{8}$$

From Formula 8, we can conclude that $N^M + N^K < N^{M*K}$, *i.e.* the complexity of the 2L scheduling approach is smaller than that of the 1L scheduling approach.

Let us assume that we have $n$ tasks to be scheduled at $m$ sites and $n >> m$. The complexity of the first loop (lines 2-5) of the DIM algorithm is $\mathcal{O}(n)$. The complexity of the $TaskReschedule$ algorithm is $\mathcal{O}(n)$, since there may be $n$ tasks scheduled at a site in the first loop (lines 2-5) of the DIM algorithm. The complexity of $MinTime(S)$ and $MaxTime(S)$ is $\mathcal{O}(m)$, which is much smaller than $\mathcal{O}(n)$. Thus, the complexity of Lines $7 - 9$ is $\mathcal{O}(n)$. Assume that the difference between the maximum total time and the minimum total time is $T_{diff}$. The maximum value of $T_{diff}$ can be $n * avg(T)$ when all the tasks are scheduled at one site while there is no task scheduled at other sites. $avg(T)$ represents the average execution time of each task, which is a constant value. After $m$ ($m^2$) times[1] of rescheduling tasks between the site of maximum total

[1] When each site has the same computing capacity, this is $m$. But when not all the sites have the same computing capacity, this should be $m^2$.

time and the site of minimum total time, the maximum difference of total time of any two sites should be reduced to less than $\frac{T_{diff}}{\sqrt{2}}$ (see Appendix). Thus, the complexity of the second loop (lines 6-9) of the DIM algorithm is $\mathcal{O}(m \cdot n \cdot \log n)$ ($\mathcal{O}(m^2 \cdot n \cdot \log n)$). Therefore, the complexity of the DIM algorithm is $\mathcal{O}(m \cdot n \cdot \log n)$ ($\mathcal{O}(m^2 \cdot n \cdot \log n)$). The complexity indicates that when the number of tasks or sites are small, the scheduling problem is simple. Since $m$ is much smaller than $n$, it is only a little bit higher than those of OLB and MCT, which is $\mathcal{O}(m \cdot n)$, but yields high reduction in SWf execution (see Section 6.3).

## 6 Experimental Evaluation

In this section, we present an experimental evaluation of our DIM scheduling algorithm using Microsoft Azure multisite cloud [3]. First, we present two real-life SWfs, *i.e.* Buzz and Montage, as use cases. Then, we explain the techniques for the implementation of intersite communication of multisite Chiron in Azure. Afterwards, we show the experimental results of executing the two SWfs in Azure with different multisite scheduling algorithms.

### 6.1 SWf Use Cases

In this section, we present two SWfs, *i.e.* Buzz and Montage, to evaluate our proposed algorithms. The two SWfs have different structures, which can show that our proposed algorithm is suitable for different SWfs.

**Buzz Workflow** Buzz workflow is a data-intensive SWf that searches for trends and measures correlations in scientific publications. It analyses data collected from bibliography databases such as the DBLP Computer Science Bibliography (DBLP) [2] or the U.S. National Institutes of Health's National Library of Medicine (PubMed) during the last 20 years. Buzz workflow is composed of thirteen activities, which are shown in Figure 8. Boxes in the figure represent SWf activities and arrows represent the data dependencies. The FileSplit activity gathers the information of scientific publications from a bibliography databases. The Buzz activity uses the gathered information to identify buzzwords, *i.e.* a word or phrase that can become popular for a specific period of time. The WordReduce activity organizes these publications according to buzzword and publication year, and it also computes occurrences of the buzzwords. Furthermore, the YearFilter activity selects buzzwords that appeared in the publications after a specific time. The BuzzHistory activity creates a history for each buzzword. The FrequencySort activity computes the frequency of each buzzword. With this information, the HistogramCreator activity generates some histograms with word frequency varying the year. On the other hand, the Top10 activity selects ten of the most frequent words in recent years, whilst the ZipfFilter activity selects terms according to a Zipf curve that is specified by word frequency values [36]. In addition, the CrossJoin activity merges results from the Top10 activity and

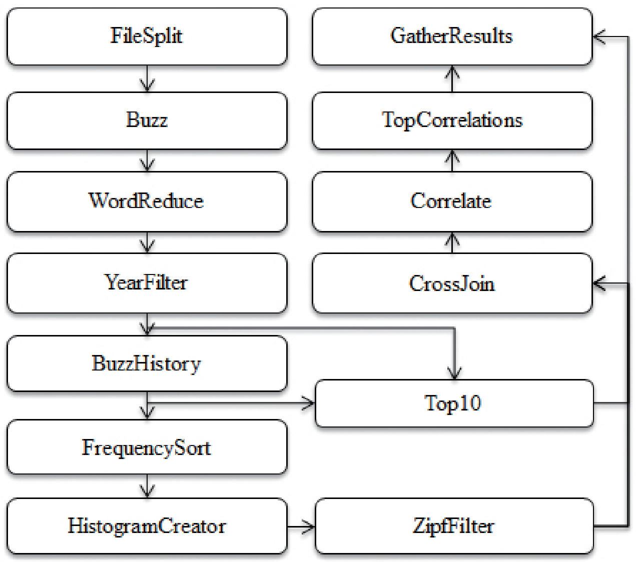

Fig. 8: **Buzz Workflow.**

the ZipfFilter activity. The Correlate activity computes correlations between the words from the Top10 activity and buzzwords from the ZipfFilter activity. Using these correlations, the TopCorrelations activity takes the terms that have a correlation greater than a threshold. Finally, the GatherResults activity presents these selected words with the histograms.

There are five activities, *i.e.* FileSplit, Buzz, BuzzHistory, HistogramCreator and Correlate, which correspond to multiple tasks. In our experiment, the tasks of the five activities are scheduled by the multisite scheduling algorithm. The other activities exploit a database management system to process data at the master site.

**Montage Workflow** Montage is a data-intensive SWf for computing mosaics of input images [14]. The input data and the intermediate data are of considerable size and require significant storage resources. However, the execution time of each task is relatively small, which can be at most a few minutes. The structure of the Montage SWf is shown in Figure 9. Activity 1, mProjectPP, reprojects single images to a specific scale. The mDiffFit activity performs a simple image difference between a single pair of overlapping images, which is generate by the mProjectPP activity. Then, the mConcatFit activity merges multiple parameter files into one file. Afterwards, mBgModel uses the image-to-image difference parameter table to interactively determine a set of corrections to apply to each image to achieve a "best" global fit. The mBackground activity removes a background from a single image. This activity takes the output data of the mProjectPP activity and that of the mBgModel activity. The mImgTbl

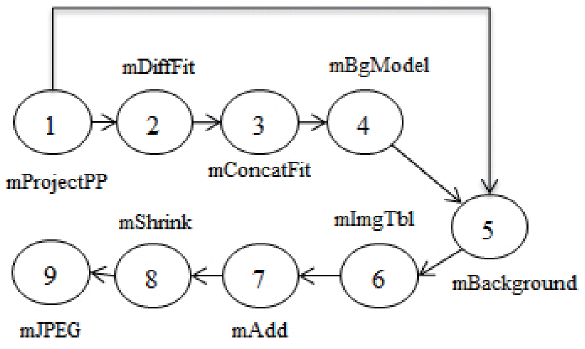

Fig. 9: **Montage Workflow.**

activity prepares the information for putting the images together. The mAdd activity generates an output mosaic and the binning of the mosaic is changed by the mShrink activity. Finally, the mJPEG activity creates a JPEG image from the mosaic. In addition, Montage can correspond to different square degrees [14] (or degree for short), which represents the size of the mosaics image. Each degree represents a certain configuration of the input data and the parameters and the lower degree corresponds to fewer input data.

There are three activities, *i.e.* mProjectPP, mDiffFit, mBackground, which correspond to multiple tasks in the Montage SWf of 0.5, 1 and 2 degree. Montage has fewer activities of multiple tasks than Buzz. However, DIM is designed for scheduling a bag of multiple tasks. Thus, the advantage of DIM is less obvious in executing Montage than Buzz, which is shown in Section 6.3.

## 6.2 Intersite Communication

In this section, we present the detailed techniques for the multisite file transfer module and multisite message communication module. We choose Azure Service Bus [1] to realize the functionality of message communication module. Azure Service Bus is a generic, cloud-based messaging system for the communication among different devices. The communication can be based on the HTTP protocol, which does not need to maintain connection information (HTTP is stateless). Although this may bring more overhead for each message, the amount of control messages is low and this cost is negligible. The file transfer module is realized by Java TCP connections between two master nodes of two different sites. Since the idle intersite TCP connections may be cut down by the cloud operator, *e.g.* every $5-10$ minutes in Azure, the connections are maintained by sending *keepalive* messages. For instance, two messages per time period. Before execution, a task is scheduled at a site by the multisite task scheduler. If they are not stored at the scheduled site, the input files of the task are transferred to the scheduled site by the multisite file transfer module.

## 6.3 Experiments

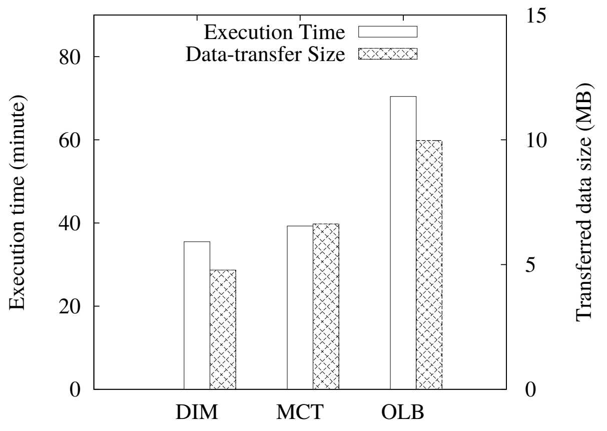

Fig. 10: **Buzz Execution time.** The amount of data is 60MB.

This section gives our experimental evaluation of the DIM algorithm, within Microsoft Azure. Azure [3] has multiple cloud sites, *e.g.* Central US (CUS), West Europe (WEU) and North Europe (NEU). We instantiated three A4 [5] (8 CPU cores) VMs at each of the three site, *i.e.* CUS, WEU and NEU. We take WEU as master site. We deploy an A2 (2 CPU cores) VM at WEU and install PostgreSQL database as provenance data. We assume that the input data of the SWfs are distributed at the three sites. We compare our proposed algorithm with two representative baseline scheduling algorithms, *i.e.* Opportunistic Load Balancing (OLB) and Minimum Completion Time (MCT). In the experiment, we assume the input data of SWfs cannot be moved. Thus, we schedule the tasks of start activities, e.g. FileSplit and mProjectPP, to where the data is while exploiting DIM, OLB or MCT to schedule tasks of the other activities. In the multisite environment, OLB randomly selects a site for a task while MCT schedules a task to the site that can finish the execution first. In the following figures, the execution time is the absolute time for SWf execution and the data-transfer size refers to the input data of tasks, *i.e.* the intermediate data transferred across different sites, which does not include the provenance data. In addition, since the resource utilization also depends on the programs used in different SWfs and that each SWf exploits various programs, we did not measure it.

First, we used a DBLP 2013 XML file of 60MB as input data for Buzz SWf in our experiments. The input data is evenly partitioned into three parts, which have almost the same amount of data. Each part is distributed and stored at a site while configuration files of Buzz SWf are present at all the three sites. We take WE as a master site to execute the Buzz workflow. The provenance database and Azure Service Bus are also located at the WE site. The execution result corresponding to each scheduling algorithm is shown in Figure 10. The execution time in Figures 10, 11, 12 and 13 represents the total execution time including data transfer and scheduling time. Table 1 shows the execution time, the time to transfer input data and the time to generate provenance data in one task of Buzz activity. This table shows that the time to transfer input data and provenance data should not be ignored compared with the time to execute the task.

Table 1: **Various time.** The unit of data is KB and the unit of time is second. "Input" and "Provenance" represent the corresponding data. The task is executed at the CUS site.

| Execution Time | Input size | Input transfer time | Provenance transfer time |
|---|---|---|---|
| 1.12 | 40 | 1.945 | 0.78 |

Figure 10 shows that DIM is much better than MCT and OLB in terms of both execution time and transferred data size. The execution time of DIM is 9.6% smaller than that of MCT and 49.6% smaller than that of OLB. The size of the data transferred between different sites with MCT is 38.7% bigger than that with DIM and the size of OLB is 108.6% bigger than that with DIM. Although OLB is a random algorithm, it distributes the tasks to each site with the same probability and the transferred data remains the same for the same configuration of the SWf and cloud environment. As a result, the size of intersite transferred data can represent the average results, i.e. which are calculated from the execution of multiple tasks.

Second, we performed an experiment using a DBLP 2013 XML file of 1.29GB as input data for Buzz SWf while configuration files of Buzz SWf are present at all the three sites. The other configuration is the same as the first one. The execution result corresponding to each scheduling algorithm is shown in Figure 11.

Figure 11 shows that the advantage of DIM in terms of both execution time and transferred data size compared with MCT and OLB increases with bigger amounts of input data. The execution time corresponding to DIM is 24.3% smaller than that with MCT and 45.9% smaller than that with OLB. The size of the data transferred between different sites with MCT is 7.19 times bigger than that with DIM and the size with OLB is 7.67 times bigger than that with DIM.

Since the DIM algorithm considers the time to transfer intersite provenance data and makes optimization for a bag of tasks, *i.e.* global optimization, it can reduce the total time. Since DIM schedules the tasks to where the input data is located at the beginning, DIM can reduce the amount of intersite transferred data compared with other algorithms. MCT only optimizes the load balancing for each individual task, *i.e.* local optimization, among different sites without consideration of the time to transfer intersite provenance data. It is a greedy algorithm that can reduce the execution time by balancing the total time of

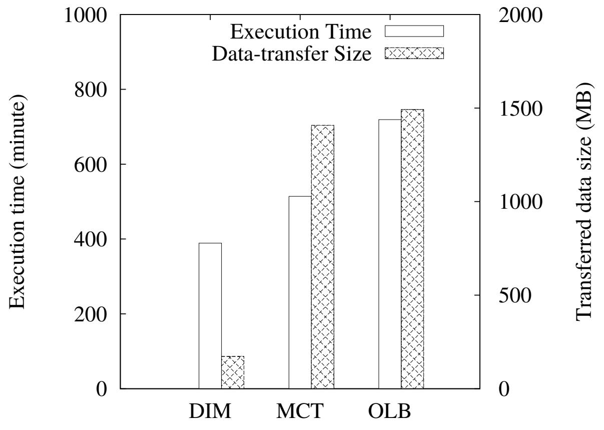

Fig. 11: **Buzz SWf Execution time.** The amount of data is 1.29GB.

each site while scheduling each task. However, it cannot optimize the scheduling for the whole execution of all the tasks of an activity. In addition, compared with OLB, MCT cannot reduce much the transferred data among different sites. Since OLB simply tries to keep all the sites working on arbitrary tasks, it has the worst performance.

Furthermore, we executed the Montage SWf with 0.5 degree with three sites, *i.e.* CUS, WEU and NEU. The size of input data is 5.5GB. The input data is evenly partitioned to three parts stored at the corresponding sites with configuration files stored at all the three sites. The execution time and amount of intersite transferred data corresponding to each scheduling algorithm are shown in Figure 12.

The execution results of Montage with 0.5 degree reveals that the execution time of DIM is 21.7% smaller than that of MCT and 37.1% smaller than that of OLB. This is expected since DIM makes optimization for a bag of tasks in order to reduce intersite transferred data with consideration of the time to transfer intersite intermediate data and provenance data. MCT is optimized for load balancing only with consideration of intermediate data. OLB has no optimization for load balancing. In addition, the intersite transferred data of DIM is 42.3% bigger than that of MCT. Since DIM is designed to achieve load balancing of each site to reduce total time, it may yield more intersite transferred data in order to achieve load balance. However, the amount of intersite transferred data of DIM is 28.6% smaller than that of OLB. This shows the efficiency of the optimization for the data transfer of DIM. Moreover, when the degree (0.5) is low, there is less data to be processed by Montage, and the number of tasks to schedule is small. Since DIM is designed for high numbers of tasks, the amounts of intersite transferred data are not reduced very much in this situation.

In addition, we executed Montage SWf of 1 degree in the multisite cloud. We used the same input data as in the previous experiment, *i.e.* 5.5GB input

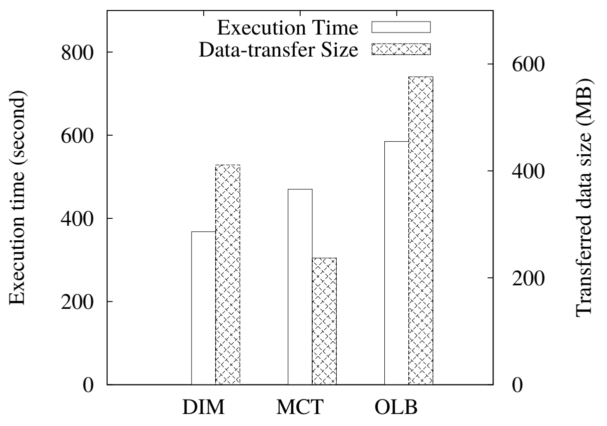

Fig. 12: **Montage SWf Execution time.** 0.5 degree.

data evenly distributed at three sites. The execution time and the amount of intersite transferred data corresponding to each scheduling algorithm are shown in Figure 13.

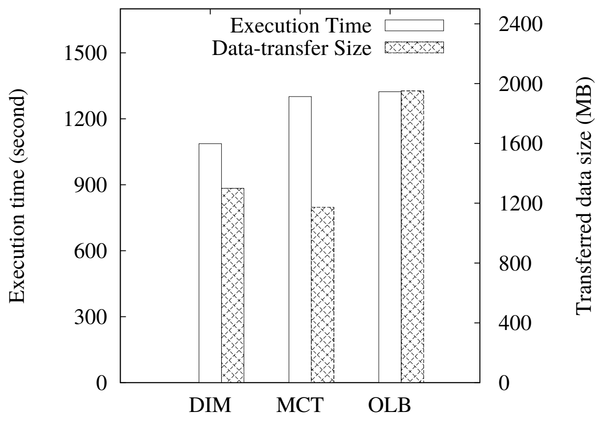

Fig. 13: **Montage SWf Execution time.** 1 degree.

The execution results of Montage with 1 degree reveals that the execution time of DIM is 16.4% smaller than that of MCT and 17.8% smaller than that of OLB. As explained before, this is expected since DIM can reduce the execution time by balancing the load among different sites compared with MCT and OLB. In addition, the intersite transferred data of DIM is 10.7% bigger than that of MCT. This is much smaller than the value for 0.5 degree (42.3%), since there are more tasks to schedule when the degree is 1 and DIM reduces intersite transferred data for a big amount of tasks. However, the amount of intersite transferred data is bigger than that of MCT. This happens since the main objective of DIM is to reduce execution time instead of reducing intersite transferred data. In addition, the amount of intersite transferred data of DIM is 33.4% smaller than that of OLB, which shows the efficiency of the optimization for the data transfer of DIM.

We also executed the Montage SWf of 2 degree in the multisite cloud. The configuration is the same as in the previous execution of Montage. The results in Figure 14 show that, although the total execution time of DIM is higher than those of MCT and OLB, the execution time without scheduling time is still smaller than those of MCT (7.0%) and OLB (12.8%). The amount of intersite transferred data of DIM is 29.2% and 61.8% smaller than those of MCT and OLB, which shows that DIM can greatly reduce intersite transferred data for scheduling big numbers of tasks. Since the cost to get information from the provenance database is expensive, the scheduling time is high when the SWf has many tasks. However, we could simply load the provenance data once in memory for scheduling and use the in-memory data to run the scheduling algorithms, which largely reduces the scheduling time. We improved the implementation of the scheduling algorithms based on this method, and executed the Montage SWf of 2 degree with the optimized version of multisite Chiron. The results are shown in Figure 15. The total execution time of DIM is 12.7% and 17.6% smaller than those of MCT and OLB.

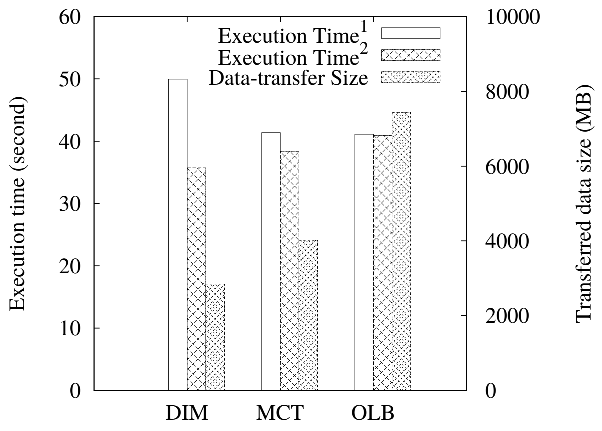

Fig. 14: **Montage Execution time.** 2 degree. Execution time$^1$ represents the total execution including the scheduling time and data transfer. Execution time$^1$ represents the execution time without scheduling time.

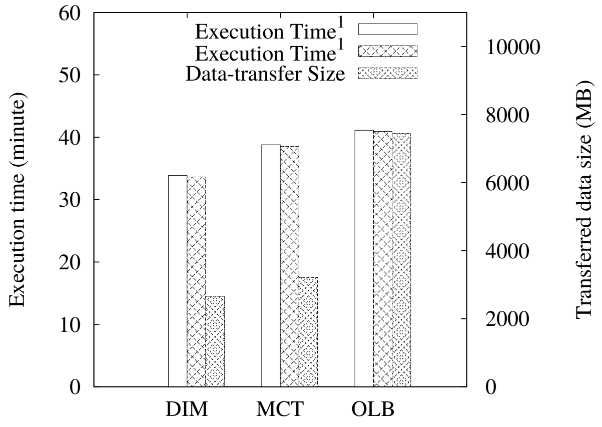

Fig. 15: **Montage Execution time.** 2 degree. Execution time$^1$ represents the total execution including the scheduling time and data transfer. Execution time$^1$ represents the execution time without scheduling time.

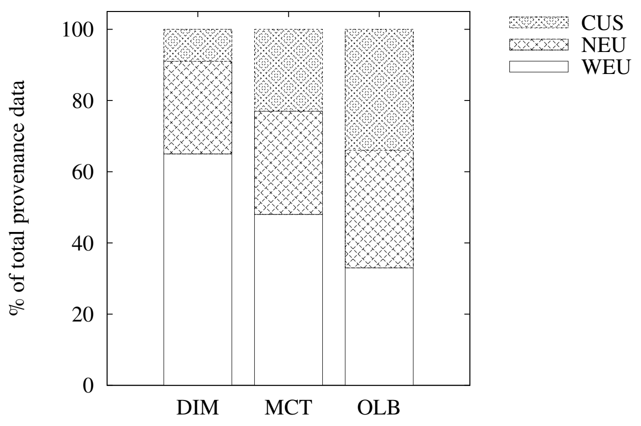

Fig. 16: **Distribution of provenance during the execution of Buzz workflow.** The size of input data is 1.2GB.

Table 2: **Scheduling Time.** The time unit is second. The size of the input data of Buzz$^{60}$ SWf is 60MB and that of Buzz$^{1.29}$ is 1.29GB. The degree of Montage$^{0.5}$ is 0.5, that of Montage$^{1}$ is 1 and that of Montage$^{2}$ is 2. "(O)" represents the scheduling time corresponding to the optimized implementation of scheduling algorithms.

| Algorithm | DIM | MCT | OLB |
|---|---|---|---|
| Buzz$^{60}$ | 71.4 | 9.8 | 1.3 |
| Buzz$^{1.29}$ | 633 | 109 | 17 |
| Montage$^{0.5}$ | 8.4 | 3.7 | 1.1 |
| Montage$^{1}$ | 29.2 | 28.8 | 1.5 |
| Montage$^{2}$ | 855.4 | 178.7 | 10.9 |
| Montage$^{2}$(O) | 15.5 | 15.1 | 10.9 |

We also measured the time to execute the scheduling algorithms to generate scheduling plans while executing Buzz and Montage. The scheduling time is shown in Table 2. The complexity of MCT is the same as that of OLB, which is $\mathcal{O}(m \cdot n)$. However, the scheduling time of MCT is much bigger than OLB (without the optimization of the implementation of scheduling algorithms). The reason is that MCT needs to interact with the provenance database to get the information of the files in order to estimate the time to transfer the files among different sites. The table shows that the time to execute DIM is much higher than OLB for both Buzz and Montage (without the optimization of the implementation of scheduling algorithms) since the complexity of DIM is higher than that of OLB and that DIM has more interactions with the provenance database in order to estimate the total time to execute the tasks at a site. When there is significant number of tasks to schedule (for the Buzz$^{1.29}$ SWf), the time to execute DIM is much bigger than that of MCT because of higher complexity and the frequent interaction with the provenance database. However, when the number of tasks is not very big, the time to execute DIM is similar to that of MCT, both of which are much bigger than that of OLB, since it takes much time to communicate with the provenance database to get the data location information for the estimation of the total time to execute tasks at each site. This overhead can be calculated according to Formula 9, where $|T|$ represents the number of tasks and $AvgTransactionTime$ is the average time for one transaction (see Formula 6). We improved the implementation of the scheduling algorithms by loading the provenance data into the memory once and using the in-memory provenance data for the scheduling algorithms. We measured the scheduling time for executing Montage of 2 degree. The results show a major improvement, *i.e.* the scheduling time of DIM is almost the same as those of MCT and OLB. The scheduling time of the three scheduling algorithms is always small compared with the total execution time, which is acceptable for task scheduling during SWf execution time. In addition, based on the measured scheduling time, we estimate that when the number of tasks is less than one million, the scheduling time of DIM with the improved implementation is less than 10% of the total execution time, which is acceptable. According to [10], an SWf of a million tasks is already much bigger than a very large SWf (1000 tasks). Although the scheduling time of DIM may be higher than those of MCT and OLB, the execution time of SWfs corresponds to DIM is much smaller than those of MCT and OLB as explained in the experiments. This means that DIM generates better scheduling plans compared with MCT and OLB.

$$EstimationOverHead(T) = |T| * AvgTransactionTime \tag{9}$$

Table 3: **Size of Provenance Data.** The unit of the data is MB. The size of the input data of Buzz SWf is 1.2GB and the degree of Montage is 1.

| Algorithm | DIM | MCT | OLB |
|---|---|---|---|
| Buzz | 301 | 280 | 279 |
| Montage | 10 | 10 | 10 |

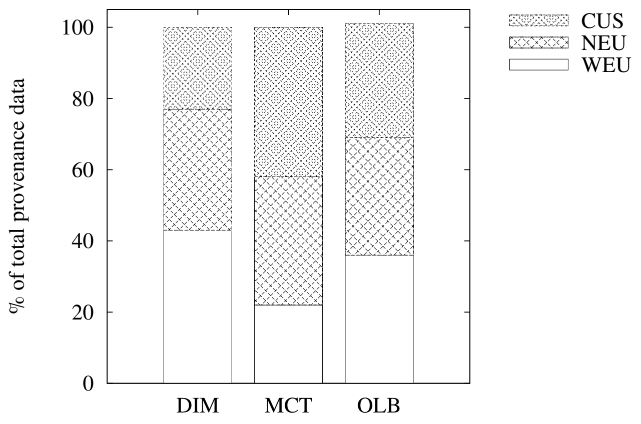

Fig. 17: **Distribution of provenance during the execution of Montage workflow.** The degree is 1.

Furthermore, we measured the size of provenance data and the distribution of the provenance data. As shown in Table 3, the amount of the provenance data corresponding to the three scheduling algorithms are similar (the difference is less than 8%). However, the distribution of the provenance data is different. In fact, the bandwidth between the provenance database and the site is in the following order: WEU > NEU > CUS [1]. As shown in Figures 16 and 17, the provenance data generated at the CUS site is much bigger than those generated at the NEU site and the WEU site for the DIM algorithm. In addition, the percentage of provenance data at WEU corresponding to DIM is much bigger than MCT (up to 95% bigger) and OLB (up to 97% bigger). This indicates that DIM can schedule tasks to the site (WEU) that has bigger bandwidth with the provenance database (the database is at the WEU site), which yields bigger percentage of provenance data generated at the site. This can reduce the time to generate provenance data in order to reduce the overall multisite execution time of SWfs. However, MCT and OLB are not aware of the provenance data transfer costs, which correspond to bigger multisite execution time. In the algorithm, we used the predefined bandwidth and execution time, which depends largely on if the user is familiar with the environment and the SWf. We leave study of the preciseness of the cost estimation as future work.

From the experiments, we can see that DIM performs better than MCT (up to 24.3%) and OLB (up to 49.6%) in terms of execution time although it takes more time to generate scheduling plans. DIM can reduce the intersite transferred data compared with MCT (up to 719%) and OLB (up to 767%). As the amount of input data increases, the advantage of DIM becomes more important.

## 7 Conclusion

In this paper, we proposed a solution based on multisite Chiron to execute a SWf using provenance data and process distributed data in a multisite cloud.

Multisite Chiron is able to execute SWfs in a multisite cloud with geographically distributed input data. We proposed the architecture of multisite Chiron, defined a new provenance model for multisite SWf execution and a global method to gather the distributed provenance data in a centralized database. Based on this architecture, we proposed a new scheduling algorithm, *i.e.* DIM, which considers the latency to transfer data and to generate provenance data in multisite cloud. We analyzed the complexity of DIM ($\mathcal{O}(m \cdot n \cdot \log n)$), which is quite acceptable for scheduling bags of tasks. We used two real-life SWfs, *i.e.* Buzz and Montage to evaluate the DIM algorithm in Microsoft Azure with three sites. The experiments show that although its complexity is higher than those of OLB and MCT, DIM is much better than two representative baseline algorithms, *i.e.* MCT (up to 24.3%) and OLB (up to 49.6%), in terms of execution time. In addition, DIM can also reduce significantly data transfer between sites, compared with MCT (up to 719%) and OLB (up to 767%). The advantage of DIM becomes important with high numbers of tasks.

[1] For instance, the time to execute "SELECT count(*) from eactivity" at the provenance database from each site: 0.0027s from the WEU site, 0.0253s from the NEU site and 0.1117s from the CUS site.

## Acknowledgment

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

## Appendix

In this section, we analyze the convergence of the DIM algorithm. Since there are finite tasks in the bag of tasks scheduled at Site $s_i$, Algorithm 2 always converges. Next, let us analyze the convergence of Algorithm 1. In order to make the problem simple, we assume that the total time at the two sites are the same after executing Algorithm 2 and we denote the time to transfer data, including input data and provenance data, as $\alpha * ExecTime(T, s)$ in Formula 3. Thus, we have:

$$\begin{aligned} TotalTime(T,s) =& (1+\alpha) * ExecTime(T,s) = C * \frac{|T|}{CC(s)} \\ C =& (1+\alpha) * AvgWorkload(T) \\ CC(s) =& \sum_{VM_i \in s} ComputingCapacity(VM_i) \end{aligned} \tag{10}$$

*TotalTime*$(T, s)$, $|T|$, *AvgWorkload*$(T)$ and *ComputingCapacity*$(VM_i)$ represent the same values as those in Formula 3. $CC(s)$ represents the computing capacity at each site. $C$ and $CC(s)$ are constant values. We denote the total execution time at each site by $T_{opt}$ when all the sites are in the optimal situation, *i.e.* each site has the same total execution time. We denote a cost function in Formula 11 to measure the distance of current scheduling plan ($SP$) and the optimal scheduling plan.

$$J(SP) = \sum_{i=1}^{m}(J(s_i)) = \sum_{i=1}^{m}(TotalTime(T_i, s_i) - T_{opt})^2 \tag{11}$$

Where $J(s_i)$ represents the cost function of Site $s_i$. $T_i$ and $s_i$ are defined by the scheduling plan $SP$.

First, let us analyze the situation when all the sites have the same computing capacity. In each iteration of Algorithm 1, we choose two sites ($s_i$, $s_j$) with the maximum total execution time and minimum total execution time. That means that we choose at least one site $s_i$, which has the biggest distance between the current situation and the optimal situation among all the sites as shown in Formula 12. In addition, the total execution time of the maximum total execution time should be bigger than or equal to $T_{opt}$ and the total execution time of the minimum total execution time should be smaller than or equal to $Topt$ as shown in Formula 13.

$$J(s_i) = \max_{j=1}^{m}(TotalTime(T_j, s_j) - T_{opt})^2 \tag{12}$$

$$(TotalTime(T_i, s_i) - T_{opt}) * (T_{opt} - TotalTime(T_j, s_j)) \geq 0 \tag{13}$$

We denote the other site by $s_j$. Since the two sites have the same computing capacity, we denote the total execution time of each site by $TotalTime'(T, s_{ij})$ after executing Algorithm 2. Since the two sites have the same execution time, according to Formula 10, they should have the same number of tasks, which is denoted by $T$. Thus, we have Formula 14.

$$\begin{aligned} TotalTime(T_i, s_i) =& C * \frac{|T_i|}{CC(s)} \\ TotalTime(T_j, s_j) =& C * \frac{|T_j|}{CC(s)} \\ TotalTime'(T, s_{ij}) =& C * \frac{|T|}{CC(s)} \\ 2 * |T| =& |T_i| + |T_j| \end{aligned} \tag{14}$$

According this formula, we can get Formula 15.

$$TotalTime'(T, s_{ij}) = \frac{TotalTime(T_i, s_i) + TotalTime(T_j, s_j)}{2} \tag{15}$$

Thus, the cost function of the two selected sites after one iteration can be expressed as Formula 16.

$$\begin{aligned} J'(SP', s_i, s_j) =& (TotalTime'(T, s_i) - T_{opt})^2 + (TotalTime'(T, s_j) - T_{opt})^2 \\ =& 2 * (TotalTime'(T, s_{ij}) - T_{opt})^2 \\ =& 2 * (\frac{TotalTime(T_i, s_i) + TotalTime(T_j, s_j)}{2} - T_{opt})^2 \\ =& 2 * (\frac{(TotalTime(T_i, s_i) - T_{opt}) - (T_{opt} - TotalTime(T_j, s_j)}{2}))^2 \\ =& \frac{(TotalTime(T_i, s_i) - T_{opt})^2 + (T_{opt} - TotalTime(T_j, s_j)^2}{2} \\ & - (TotalTime(T_i, s_i) - T_{opt}) * (T_{opt} - TotalTime(T_j, s_j) \\ =& \frac{J(SP, s_i, s_j)}{2} \\ & - (TotalTime(T_i, s_i) - T_{opt}) * (T_{opt} - TotalTime(T_j, s_j) \end{aligned} \tag{16}$$

Where we denote the scheduling plan after the iteration by $SP$'. We denote the cost function of the two selected sites before the iteration as $J(SP, s_i, s_j)$. According to Formula 13, we get the Formula 17.

$$J'(SP', s_i, s_j) < \frac{J(SP, s_i, s_j)}{2} \tag{17}$$

Thus, after $m$ iterations, $J(SP)$ becomes less than $\frac{J(SP)}{2}$. The minimum modification in one iteration should be bigger than $\frac{C}{CC(s)}$. Thus, after at most $m * \log_2(J(SP) - \frac{C}{CC(s)})$ iterations, Algorithm 1 terminates.

Then, let us consider a situation where some sites ($\rho m$, $> 1\rho > 0$) have much bigger computing capacity than other sites ($(1-\rho)m$). We assume that after executing Algorithm 2 between two sites of different computing capacity, the total execution time of the two sites becomes the original total execution time of the site, which has bigger computing capacity. In this situation, in order to reduce $J(SP)$ to $\frac{J(SP)}{2}$, we need at most $\rho * (1-\rho) * m^2$. Thus, after at most $\rho * (1-\rho) * m^2 * \log_2(J(SP) - \frac{C}{CC(s)})$ iterations, Algorithm 1 terminates. Furthermore, the other situations are between the first situation where all the sites have the same computing capacity and this situation.

## References

1. Azure service bus. `http://azure.microsoft.com/en-us/services/service-bus/`.
2. DBLP Computer Science Bibliography. `http://dblp.uni-trier.de/`.
3. Microsoft Azure. `http://azure.microsoft.com`.
4. Montage. `http://montage.ipac.caltech.edu/docs/gridtools.html`.
5. Parameters of different types of vms in microsoft Azure. `https://azure.microsoft.com/en-us/pricing/details/virtual-machines/`.
6. K. Bhuvaneshwar, D. Sulakhe, R. Gauba, A. Rodriguez, R. Madduri, U. Dave, L. Lacinski, I. Foster, Y. Gusev, and S. Madhavan. A case study for cloud based high throughput analysis of {NGS} data using the globus genomics system. *Computational and Structural Biotechnology Journal*, 13:64 – 74, 2015.
7. L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez. Dynamic query scheduling in data integration systems. In *Proceedings of the 16th Int. Conf. on Data Engineering*, pages 425–434, 2000.
8. L. Bouganim, O. Kapitskaia, and P. Valduriez. Memory-adaptive scheduling for large query execution. In *Proceedings of the 1998 ACM CIKM Int. Conf. on Information and Knowledge Management*, pages 105–115, 1998.
9. J. Cala, Y. Xu, E.A. Wijaya, and P. Missier. From scripted hpc-based NGS pipelines to workflows on the cloud. In *14th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 694–700, 2014.
10. R. N. Calheiros and R. Buyya. Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Transactions on Parallel and Distributed Systems*, 25(7):1787–1796, 2014.
11. 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.

12. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150, 2004.
13. 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.
14. E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The montage example. In *Int. Conf. for High Performance Computing, Networking, Storage and Analysis.*, pages 1–12, 2008.
15. E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.
16. J. Dias, E. S. Ogasawara, D. de Oliveira, F. Porto, P. Valduriez, and M. Mattoso. Algebraic dataflows for big data analysis. In *IEEE Int. Conf. on Big Data*, pages 150–155, 2013.
17. 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.
18. 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.
19. H. Hiden, P. Watson, S. Woodman, and D. Leahy. e-science central: cloud-based e-science and its application to chemical property modelling. Tchnical Report CS-TR-1227, 2010.
20. H. Hiden, S. Woodman, P. Watson, and J. Cala. Developing cloud applications using the e-science central platform. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 371(1983), 2012.
21. J. Liu, E. Pacitti, P. Valduriez, D. de Oliveira, and M. Mattoso. Multi-objective scheduling of scientific workflows in multisite clouds. *Future Generation Computer Systems*, 63:76–95, 2016.
22. J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. Parallelization of scientific workflows in the cloud. Research Report RR-8565, 2014.
23. 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.
24. J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. Scientific Workflow Scheduling with Provenance Support in Multisite Cloud. In *12th Int. Meeting on High Performance Computing for Computational Science VECPAR*, page 8, 2016.
25. 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.
26. 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.
27. V. Martins, E. Pacitti, M. E. Dick, and R. Jiménez-Peris. Scalable and topology-aware reconciliation on P2P networks. *Distributed and Parallel Databases*, 24(1-3):1–43, 2008.
28. M. Mattoso, J. Dias, K. A. C. S. Oca na, E. Ogasawara, F. Costa, F. Horta, V. Silva, and D. de Oliveira. Dynamic steering of HPC scientific workflows: A survey. *Future Generation Computer Systems*, 2014.

29. 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.
30. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems.* Springer, 2011.
31. E. Pacitti, R. Akbarinia, and M. E. Dick. *P2P Techniques for Decentralized Applications.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
32. L. Pineda-Morales, A. Costan, and G. Antoniu. Towards multi-site metadata management for geographically distributed cloud workflows. In *2015 IEEE Int. Conf. on Cluster Computing, CLUSTER*, pages 294–303, 2015.
33. R. Sandberg, D. Golgberg, S. Kleiman, D. Walsh, and B. Lyon. Innovations in internetworking. chapter Design and Implementation of the Sun Network Filesystem, pages 379–390. 1988.
34. O. Schenk and K. Gärtner. Two-level dynamic scheduling in PARDISO: improved scalability on shared memory multiprocessing systems. *Parallel Computing*, 28(2):187–197, 2002.
35. 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.
36. K. Tarapanoff, L. Quoniam, R. H. de Araújo Júnior, and L. Alvares. Intelligence obtained by applying data mining to a database of french theses on the subject of brazil. *Information Research*, 7(1), 2001.
37. 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.
38. M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Record*, 34(3):56–62, 2005.
39. 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.