CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **25**:2523–2539
Published online 4 July 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3084

SPECIAL ISSUE PAPER

A clustering based coscheduling strategy for efficient scientific workflow execution in cloud computing

Kefeng Deng[1,*], Kaijun Ren[1], Junqiang Song[1], Dong Yuan[2], Yang Xiang[3] and Jinjun Chen[4]

[1]*School of Computer, National University of Defense Technology, Changsha, Hunan 410073, P.R. China*
[2]*Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Victoria 3122, Australia*
[3]*School of Information Technology, Deakin University, 221 Burwood Highway, Burwood, Vic 3125, Australia*
[4]*Faculty of Engineering and Information Technology, University of Technology, Sydney, PO Box 123, Broadway NSW 2007, Australia*

SUMMARY

Due to its advantages of cost-effectiveness, on-demand provisioning and easy for sharing, cloud computing has grown in popularity with the research community for deploying scientific applications such as workflows. Although such interests continue growing and scientific workflows are widely deployed in collaborative cloud environments that consist of a number of data centers, there is an urgent need for exploiting strategies which can place application datasets across globally distributed data centers and schedule tasks according to the data layout to reduce both latency and makespan for workflow execution. In this paper, by utilizing dependencies among datasets and tasks, we propose an efficient data and task coscheduling strategy that can place input datasets in a load balance way and meanwhile, group the mostly related datasets and tasks together. Moreover, data staging is used to overlap task execution with data transmission in order to shorten the start time of tasks. We build a simulation environment on Tianhe supercomputer for evaluating the proposed strategy and run simulations by random and realistic workflows. The results demonstrate that the proposed strategy can effectively improve scheduling performance while reducing the total volume of data transfer across data centers. Concurrency and Computation: Practice and Experience, 2013.© 2013 Wiley Periodicals, Inc.

Received 16 January 2013; Accepted 4 June 2013

KEY WORDS: cloud computing; scientific workflow; coscheduling; data placement; data staging

1. INTRODUCTION

In modern scientific collaboration, scientific workflows are quickly becoming a popular and important mechanism by helping scientists automate the processes of scientific simulation and data analysis which are conventionally time-consuming and error-prone [1]. In many research fields, especially in interdisciplinary fields such as bioinformatics and climate simulation, scientific workflows are usually both computational and data-intensive because they often consist of hundreds or thousands of tasks, consume gigabytes or terabytes input datasets, and generate similar amounts of intermediate data. Therefore, it is common for scientists to execute these workflows in distributed large-scale computational environments such as clusters and grids. However, building such systems is extremely expensive. Although some systems provide resources for free, they follow a best-effort

*Correspondence to: Kefeng Deng, *School of Computer*, *National University of Defense Technology*, *Changsha*, *Hunan 410073*, *P.R. China*.
E-mail: dengkefeng@nudt.edu.cn

way. Moreover, these systems usually belong to particular institutions, and users have to apply for resources in advance (e.g., a week or longer), which may cause a delay for getting results in urgent cases.

Recently, cloud computing has emerged as a promising platform for executing scientific workflows. Under this new paradigm, the computing and storage resources are now delivered as a kind of utility charged in a pay-as-you-go manner to the general public. Based on economies of scale and extremely large-scale commodity-computer data centers, cloud computing offers users with an illusion of infinite resources and lowers the costs to a considerable degree in the meantime. Furthermore, through virtualization technologies and widespread adoption of service computing, scientists using cloud computing are provided with an abstraction of the underlying infrastructure and consequently a highly flexible way to customize their own runtime environment for special requirements in their researches. Experiments of performing scientific applications in the clouds have been conducted [2–5]. The results demonstrate that cloud computing is a good choice for scientists who need resources instantly [5], and the costs of doing science in the clouds can be significantly reduced by providing the right amount of resources [4].

Even though a data center can provide enormous computing and storage resources for workflow execution, there are many reasons that scientific workflows have to be conducted among a number of data centers. For instance, multiple organizations may setup a collaborative computing environment that contains a number of geographically distributed data centers for executing data-intensive scientific workflows. A straightforward heuristic for executing scientific workflows in such an environment is that scientists upload their original datasets to one data center and starts the workflow, and then the rest of the datasets are dynamically retrieved from other data centers during execution. However, this heuristic ignores the important feature of data sharing in scientific collaborations. When other scientists need the newly uploaded data, they also need to retrieve the whole datasets, which will evidently result in a large volume of data transfer among different data centers and heavily affect the performance of the workflows. Moreover, executing scientific workflows in one data center will inevitably lead to highly skewed data center storage utilization because some of the scientific workflows may consume orders of magnitude datasets than others.

In this paper, we propose a *k*-means clustering based coscheduling strategy for efficient execution of scientific workflows in collaborative cloud environments. The strategy can place the input datasets in a load balance manner among geographically distributed data centers and intelligently schedule the tasks according to the layout of the datasets to achieve high performance. Our methods are summarized in the following: (i) we define three types of dependencies, which are data–data dependency, data–task dependency, and task–task dependency, to capture the correlations between datasets and tasks in a workflow; (ii) we extend our previous *k*-means clustering data placement strategy with task scheduling to intelligently place the most related datasets and tasks together [6,7]; (iii) we use a node adjustment algorithm and the task duplication technique to correct inappropriate clustering and trade computation for communication; and (iv) we use data staging to retrieve input data in advance in order to mask task execution with data transmission and to shorten the start time of the tasks. Finally, in comparison with other representative strategies under both random generated workflows and real world workflows, the experimental results show that our strategy is able to considerably reduce the volume of data transfer among different data centers and the makespan of the workflows at the same time.

The remainder of this paper is organized as follows. Section 2 briefly describes our prior *k*-means data placement strategy and its shortcomings and analyzes the research problem. In section 3, three types of dependencies are defined, and the weighted *k*-means clustering based coscheduling strategy is proposed. In section 4, we present the architecture of the SwinDeW-C (**Swin**burne **De**centralised **W**orkflow for **C**loud) scientific workflow management system. Section 5 demonstrates the efficiency of our strategy by both random workflows and realistic workflows. The related work is discussed in section 6. In section 7, we conclude the paper and present possible future work.

2. MOTIVATION AND PROBLEM ANALYSIS

2.1. Motivation

In our previous work [6], a novel matrix-based *k*-means clustering strategy was proposed for data placement while performing data-intensive scientific workflows in collaborative cloud environments. The strategy consists of two clustering algorithms which are used in workflow build-time stage and runtime stage, respectively. In order to execute a workflow, the build-time stage algorithm constructs a dependency matrix (DM) for all the application datasets based on their data dependencies. The dependency between datasets d_i and d_j is defined by the number of tasks that use both d_i and d_j: $dependency_{ij} = |T_i \cap T_j|$, where T_i is the set of tasks that take d_i as input, and T_j is the set of tasks that consume d_j. After that, the strategy clusters the matrix by using the bond energy algorithm [8] and divides the clustered matrix (CM) into *k* partitions. Then, the partitions are distributed into *k* data centers. In the situation that a partition is too large to be fit into a data center, it will be recursively partitioned into smaller ones. At runtime, the strategy calculates the dependencies to each data center for the newly generated datasets and moves the data to the data center that has the highest dependency value. Experimental results showed that the strategy can effectively reduce the number of data movements among different data centers.

However, because different datasets have different sizes, moving a small dataset is better than moving a large one, and transferring several small datasets may be more effective than transferring a large one. As such, our previous strategy suffers from the deficiency of not considering data sizes. Moreover, it is not sufficient to just consider the number of dependencies and the size of the original datasets. The intermediate datasets generated by the tasks also have an important impact on the total amount of data transfer. For one thing, the intermediate datasets have dependencies with the original datasets and other intermediate datasets as well, which will result in additional data movements when the dependent datasets are in different data centers. For another, the data movements caused by the dependencies of intermediate datasets sometimes are so large that they increase the total volume of data transfer more than the reduction gained by just considering the size of the input datasets. For example, in the case when the generated datasets are much larger than input datasets, carefully scheduling the corresponding task is essential to reduce the volume of transfer in subsequent execution.

For efficient execution of data-intensive scientific workflows, both the data placement job and the task scheduling job must be reasonably planned. Most of the recently proposed schedulers decouple data placement with task scheduling and deal with the jobs one by one. However, these jobs are inherently interrelated because the data distribution formed by initial data placement determines the decision of task scheduling, and the intermediate datasets generated by the tasks must be carefully placed into the data centers in order to form a good data distribution so as to facilitate subsequent task execution. Therefore, these jobs must be considered together to obtain satisfactory solutions, and we attempt to address the problem by proposing a task and data coscheduling strategy.

2.2. Problem analysis

In this paper, we consider the problem of running data-intensive scientific workflows in a collaborative cloud environment that consists of *k* geographically distributed data centers: $DC = \{S_1, S_2, \ldots, S_k\}$. A scientific workflow is modeled as a directed acyclic graph (DAG), $G = (V \leftarrow D \cup T,\ E,\ \omega,\ c)$, where V is the set of nodes and E is the set of edges representing the data transfer between corresponding nodes. A node in the DAG either represents a task or a dataset. An edge $e_{ij} \in E$ represents the data movement from node n_i to node n_j, where $n_i, n_j \in V$. The positive real weight associated with node $n_i \in V$ represents the size of input dataset whereas $n_i \in D$ or the size of intermediate dataset generated by some task while $n_i \in T$, which is denoted by $\omega(n_i)$. The positive real weight $c(e_{ij})$ of edge $e_{ij} \in E$ corresponds to the volume of data transfer from node n_i to node n_j, representing the communication cost and also representing a kind of data dependency between the nodes. Note that data transfer is only required when dependent nodes are distributed in different data centers.

Before executing a scientific workflow, the input datasets must be uploaded to the data centers beforehand. Traditional scientific workflow schedulers consider data resources as second-class citizens and take access to data as a side effect of computation. This may be true when workflows are executed in clusters or grids that are connected by low latency networks. However, the data centers in cloud computing environments are connected by a WAN, which will undoubtedly lead to significant latency when remote access to large datasets frequently occurs. In our previous work, the input datasets are clustered by their dependencies and distributed into different data centers according to their storage limits. In this paper, we relax the constraint and assume that each data center can provide sufficient storage during execution. However, the input data must be distributed in a load balance way in order to facilitate scientific data sharing and to avoid skewed data center storage utilization. Under the previous conditions, we try to seek for efficient strategies that will reduce the data transfer among different data centers as much as possible.

Given a scientific workflow, the entire execution process consists of two stages, that is, build-time stage and runtime stage. At build-time stage, the user's datasets are uploaded and distributed into geographically dispersed data centers according to appropriate strategies or heuristics. For example, some datasets that are always used together should be placed in the same data center. During runtime stage when a task is determined to be scheduled on a data center, it must first retrieve the input datasets that locate in other data centers to its home site before executing. The intermediate datasets generated by the task may be retrieved by subsequent tasks as well. Hence, appropriate data placement and task scheduling together is essential for obtaining good performance in scientific workflow execution. In the following sections, we will first describe our build-time and runtime coscheduling strategies. Then, the implementation and execution detail will be presented.

3. WEIGHTED *K*-MEANS CLUSTERING STRATEGY

3.1. Definition of dependencies

In this paper, we extend our previous definition of data dependency with data size and enhance the strategy with two other types of dependencies, which are dependency between tasks and dependency between data and tasks. All the three types of dependencies are shown in Figure 1, and their definitions are presented in Table I.

As shown by Figure 1(a), two datasets are dependent with each other if they are both used by the same task, and we call this type of dependency as data–data dependency. The weight of the

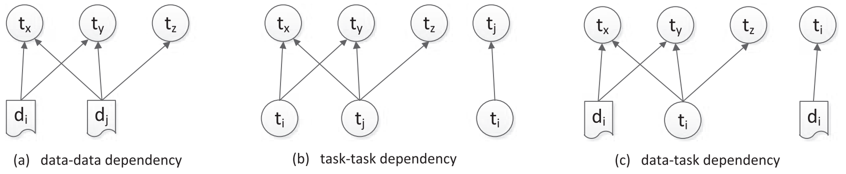


Figure 1. Three types of dependencies.

Table I. Definition of dependencies.

Type	Definition	
Data–data	$\text{dependency}_{ij} = \lvert T_i \cap T_j \rvert * (\omega(d_i) + \omega(d_j))/2$	(1)
Task–task	$\text{dependency}_{ij} = \begin{cases} (\lvert T_i \cap T_j \rvert + 1) * (\omega(t_i) + \omega(t_j))/2, & \textit{if } t_i \in T_j \textit{ or } t_j \in T_i; \\ \lvert T_i \cap T_j \rvert * (\omega(t_i) + \omega(t_j))/2, & \textit{otherwise.} \end{cases}$	(2)
Data–task	$\text{dependency}_{ij} = \begin{cases} (\lvert T_i \cap T_j \rvert + 1) * (\omega(d_i) + \omega(t_j))/2, & \textit{if } t_j \in T_i; \\ \lvert T_i \cap T_j \rvert * (\omega(d_i) + \omega(t_j))/2, & \textit{otherwise.} \end{cases}$	(3)

dependency between two datasets d_i and d_j is calculated by multiplying the number of common tasks $|T_i \cap T_j|$ and their average weight $(\omega(d_i)+\omega(d_j))/2$, which is shown by the first formula in Table I. Figure 1(b) shows another type of dependency which is named as task–task dependency. Obviously, there is a dependency between task t_i and task t_j if t_i is a parent of t_j or vice versa. Moreover, similar to the definition of data–data dependency, we say that two tasks are also dependent with each other if they are the common parents of some other tasks. Hence, the degree of dependency between two tasks t_i and t_j is $|T_i \cap T_j|+1$ if $t_i \in T_j$ or $t_j \in T_{i,}$ and the weight is defined by multiplying the degree with their average output data sizes. The definition of the third type of dependency, that is, data–task dependency is similar to the task–task dependency, and the formula is shown in the lower right corner of Table I.

3.2. *Coscheduling using k*-means clustering

Step 1: prescheduling

The weighted *k*-means clustering based coscheduling strategy is based on the aforementioned dependencies. Furthermore, the datasets and tasks will be grouped and prescheduled together into different data centers. Similar to our previous *k*-means clustering strategy, the proposed strategy also consists of two steps. First, we calculate the weight of the dependencies between all the datasets and tasks according to the definitions in Table I. And then, we setup a DM where element DM_{ij} is set as $dependency_{ij}$ (shown in Figure 2, InitialCoscheduling, lines 1–3). The DM is an $n \times n$ symmetrical matrix where n is the total number of datasets and tasks. After that, we utilize the bond energy algorithm [8] to cluster the DM which permutes the rows and columns such that the elements with high dependencies are grouped together.

Given the CM, the next step is to partition it and assign the datasets and tasks with each partition into an appropriated data center. To balance the load during the assignment of datasets and tasks, each data center should be applied with a storage constraint, which is the average

```
Function: InitialCoscheduling
Input: task list taskList, dataset list dataList, datacenter list dcList
 1  FOR $n_i \in$ taskList ∪ dataList DO
 2      For $n_j \in$ taskList ∪ dataList DO
 3          DM[i][j] = Dependency($n_i, n_j$);
 4  CM = BEA($DM$);
 5  FOR each datacenter dc ∈ dcList DO
 6      dc.load = TotalVolumeOfInputData / dcList.Size;
 7  PartitionAndAssign (CM, dcList);
 8  RETURN;
```

```
Procedure: PartitionAndAssign
Input: clustered matrix CM, datacenter list dcList
 1  dc = $\min_{j=0}^{K}(DCList[j].load * \mu >= weight(CM))$;
 2  IF dc != null DO
 3      Assign CM to dc;
 4      dc.load −= weight(CM);
 5      RETURN;
 6  ENDIF;
 7  Partition CM into $CM_T$ and $CM_B$;
 8  PartitionAndAssign ($CM_T$, dcList);
 9  PartitionAndAssign ($CM_B$, dcList);
10  RETURN;
```

Figure 2. Initial coscheduling algorithm.

volume of the input datasets. However, instead of restricting the load balance, we set an imbalance ratio μ for every data center which is 10% in the paper. During the selection of data centers, the best one for a partition is the one with the highest dependencies. In cases that a partition contains too many datasets to be fit into a data center, the partition will be recursively cut into smaller ones. The point selected for partition is the one which maximizes the equation:

$$\mathrm{PM} = \sum_{i=1}^{p}\sum_{j=1}^{p}\mathrm{CM}_{ij} * \sum_{i=p+1}^{n}\sum_{j=p+1}^{n}\mathrm{CM}_{ij} - \left(\sum_{i=1}^{p}\sum_{j=p+1}^{n}\mathrm{CM}_{ij}\right)^{2}$$

The procedure for assigning and partitioning the CM is shown in the bottom of Figure 2 by the procedure PartitionAndAssign.

Even though our strategy can group the datasets and tasks with the highest dependencies together in most cases, there may be some inappropriate placement because the clustering strategy is somewhat coarse grained. As such, we develop a node (representing a dataset or a task) adjustment algorithm to select a better data center for the inappropriately placed datasets and tasks. Another important technique is task duplication [9,10]. When a task consumes comparatively small volume of datasets but produces large volume of intermediate datasets, it may be better to duplicate the task in another data center instead of transferring the result. Next, we describe the two techniques in detail.

Step 2: node adjustment

During the process of node adjustment, we first check each dataset or task within a data center dc_h to verify whether it is beneficial to be moved outside. For a dataset, namely d_i, it may be used by some tasks in dc_h ,or it may not be used by any task in dc_h. In the first situation, moving d_i outside at this moment is not advisable because it must be transferred into dc_h again when the tasks need it. However, the algorithm may sporadically place d_i alone in dc_h such that redistributing d_i into another data center that will use it can reduce the data volume for transferring d_i. Figure 3(a) shows an example of dataset adjustment. For task t_i inside data center dc_h, the process of verifying the benefit of rescheduling t_i out of dc_h is a little complicated. As shown by Figure 3(b), task t_i not only consumes original datasets $\{\mathrm{d}_{i1},\mathrm{d}_{i2},\ldots\}$ inside but also needs datasets $\{\mathrm{d}_{j1},\mathrm{d}_{j2},\ldots\}$ outside. Meanwhile, it uses intermediate datasets generated by tasks both inside and outside, that are denoted by $\{\mathrm{t}_{i1},\mathrm{t}_{i2},\ldots\}$ and $\{\mathrm{t}_{j1},\mathrm{t}_{j2},\ldots\}$, respectively. After that, it also produces intermediate datasets which will be used by tasks inside and outside, that is, $\{\mathrm{t}'_{i1},\mathrm{t}'_{i2},\ldots\}$ and $\{\mathrm{t}'_{j1},\mathrm{t}'_{j2},\ldots\}$. In this case, we calculate the volume of data transfer both inside and outside while t_i is scheduled in data center dc_h:

$$TR_{in} = \textstyle\sum_{l=j1,j2,\ldots}\omega(d_l) + \sum_{l=j1,j2,\ldots}\omega(t_l) + \sum_{l=j1,j2,\ldots}\omega(t'_l);$$

$$TR_{out} = \textstyle\sum_{l=i1,i2,\ldots}\omega(d_l) + \sum_{l=i1,i2,\ldots}\omega(t_l) + \sum_{l=i1,i2,\ldots}\omega(t'_l);$$

Therefore, if TR_{out} is larger than TR_{in}, it is more beneficial to place t_i outside. When a dataset or a task is decided to be redistributed outside, the next step is to select an appropriate data center for it.

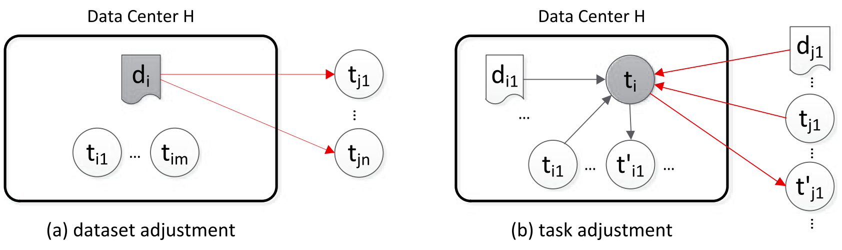


Figure 3. Node (data node or task node) adjustment.

For a dataset, if it is consumed by more than one task in a data center, we can transfer it to the data center once and buffer it until no task will need it any more. Hence, the choice for placing a dataset is straightforward, and any data center in which there are tasks that consume it is suitable. For a task, we first calculate the volume of communication between the task and each of the other data centers and then select the one that has the largest volume of data transfer with it as the target data center.

Step 3: task duplication

In this paper, we also consider scheduling in which task duplication is permitted. Task duplication means scheduling a data-intensive scientific workflow by redundantly cloning some of its tasks to other data centers. This technique is used to trade computation cost for communication cost in order to reduce the total volume of data transfer across different data centers. Because data is transferred through WANs, it may reduce the makespan of a workflow significantly. Figure 4 shows an example of task duplication. As shown by the graph, transferring the intermediate dataset generated by task t_3 will cause a communication cost by three. However, if t_3 is duplicated to data center 2, then the total volume of data transfer is only one, which results in a reduction by two.

Thus, for the tasks that use relatively small input datasets but generate large volumes of output data, it is more efficient to duplicate them to the destination data centers and recalculate the result when it is needed instead of transferring the result. There are two strategies for task duplication. The first one duplicates only the parent tasks, whereas the other attempts to duplicate the ancestor tasks as well. In this paper, the first strategy is selected because the second one is computational complex and will introduce very large number of duplicated tasks into a workflow.

Although recomputation as a replacement for data transmission may reduce the communication cost among data centers, it may introduce other side effects as well. The obvious one is computation cost. In cloud computing paradigm, everything is charged with a cost. To benefit from task duplication, we should make the reduced communication cost larger than the cost for recomputation, that is, $C_1 \times \text{Transmission} < C_2 \times \text{CPU\_Time}$ or $\frac{\text{Transmission}}{\text{CPU\_Time}} < \frac{C_2}{C_1}$, where C_1 is the unit cost for data transfer, and C_2 is the unit cost for CPU hour. Another side effect is the computation time. If one prefers to reduce the computation time, she/he should make the recomputation time less than the time for communication. This trade-off can be represented by $\text{CPU\_Time} < \frac{\text{Transmission}}{\text{Bandwidth}}$. Here, we assume that the bandwidth between two data centers is stable during the execution of a workflow. In the proposed strategy, we choose to reduce the computation time for whether a task should be duplicated or not.

4. IMPLEMENTATION

We implement the aforementioned algorithms in SwinDeW-C. The system architecture is shown in Figure 5. SwinDeW-C is developed based on SwinDeW-G (**Swin**burne **De**centralised **W**orkflow for **G**rid) [11] and is designed for the execution of large-scale cloud applications. As shown by the figure, there are two kinds of peers in SwinDeW-C, which are ordinary peer and coordinator peer. An ordinary SwinDeW-C peer inherits most components of SwinDeW-G peer including task

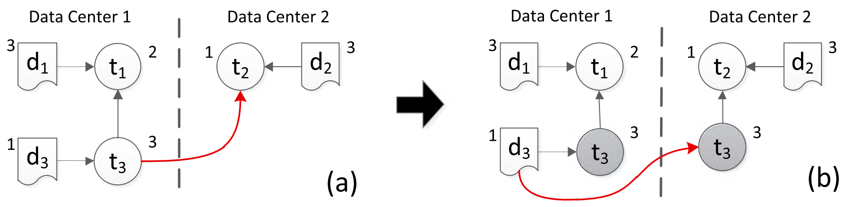


Figure 4. An example of task duplication.

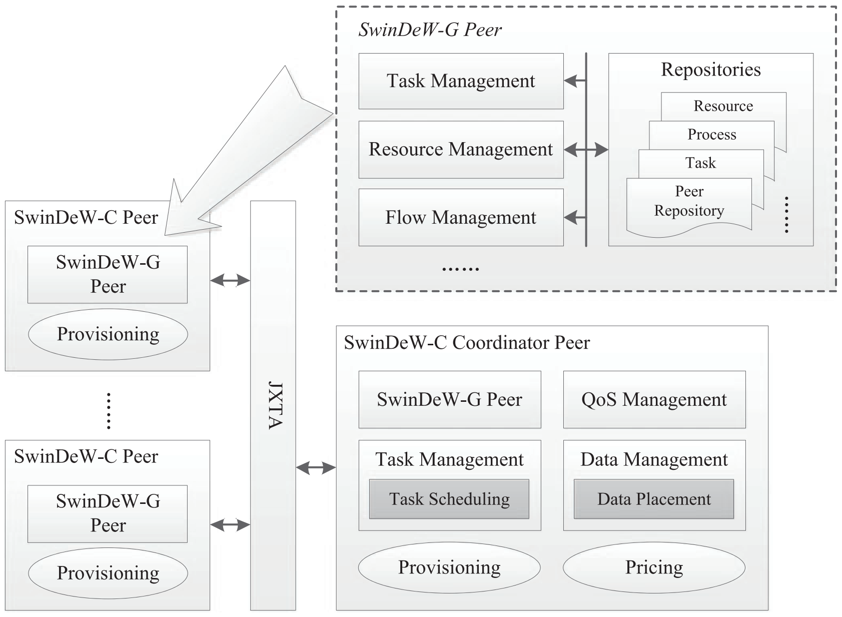


Figure 5. System architecture of SwinDeW-C peers.

management, flow management, resource management, and repositories. Unlike SwinDeW-G peer, a SwinDeW-C peer is deployed on virtual machines with scalable resources. The coordinator peers are especially developed for cloud platforms. Besides the functionality of SwinDeW-G peer, SwinDeW-C coordinator peers are equipped with additional components for scientific workflow management. For instance, the coordinator peer uses the provisioning and pricing components to make resource plans for all the peers that reside in the same data center. Particularly, a data management component is introduced in the coordinator peer for placing, storing, locating, replicating, and transferring application data. Here, we do not go into the details of these components but focus on the task scheduling and data placement modules.

4.1. *Coordination of workflow execution*

In cases when the workflows are going to be performed in a collaborative cloud environment that are composed of multiple data centers, each data center is setup with a coordinator peer for coordination of the workflow execution. Once the workflow specification is created, it will be submitted to any of the coordinator peers. At build-time stage, the selected peer runs the coscheduling algorithm that decides which task should be duplicated and which data center a task or a dataset should be distributed. After the plan is made, the input datasets are uploaded to corresponding data centers, and the partial workflow specifications are sent to appropriate coordinator peers. In this way, the whole workflow specification is mapped to different data centers, and an instance is now successfully instantiated.

During runtime stage, every task in the workflow is executed by a SwinDeW-C peer. Once the input datasets of a task are ready, the coordinator peer will dispatch the task, and the task will start execution immediately. In situations when some input data are outside the data center, the peer running the task will query the coordinator peer for the location of the data, which will then be retrieved from the remote data center. After the task gets finished, the generated data will be temporarily stored locally. The ready information will be reported to the outside coordinator peers along with its location. Hence, coordinator peers in other data centers can dispatch tasks that are waiting for the data right away. Although most of the generated data are temporal, some of them may be valuable in that they can be used by other workflows directly. In this scenario, we need to choose a suitable data center to store the data.

Similar to our previous approach, we select the data center that has the highest dependency with the dataset. Suppose the generated dataset is d_u. Firstly, we calculate the dependencies between d_u and all other datasets in the system by our previous definition: $dependency_{ui} = |T_u \cap T_i|$, for $i = 1, 2, \ldots, n$. The reason we use the old formula is that it reflects the popularity of one dataset, whereas the definitions in this work focus on minimizing the data transfer among the data centers. After that, we calculate the dependency between d_u and every data center: $dc\_dep_{uj} = \sum_{d_m \in dc_j} dependency_{um}$, for $j = 1, 2, \ldots, k$. Finally, we choose the data center with the highest value as the target data center. However, sometimes the intermediate datasets are very large and are seldom used. Therefore, we timely check the available storage space and delete the temporal data with small dependency values in each data center.

4.2. *Coordinated data staging*

Data staging is a useful technique to reduce the latency caused by data transmission. Figure 6 shows how data staging can help the coordinator peer start a task as soon as possible. As shown by subgraph (1), task t_2 needs input data d_1 and d_{t1} to start execution. In our previous method, when both datasets are ready, the coordinator peer will initiate a virtual machine and start retrieving the data. Hence, the virtual machine will begin at $t_{Stage2} = t_{Start1} + t_{E1}$, and task t_2 will start at $t_{Start2} = t_{Start1} + t_{E1} + t_{D1}$, which is illustrated by subgraph (2). Subgraph (4) shows a distinct situation where data staging begins before execution of task t_1. In this situation, the virtual machine begins at $t_{Stage2} < t_{Start1}$, and task t_2 will start at $t_{Start2} = t_{Start1} + t_{E1} + t_{G1} < t_{Start1} + t_{D1}$. That is to say the time spent on task t_1 is ideally masked by the transmission of dataset d_1.

However, it is not always helpful to start data staging too early. Subgraph (3) and (5) illustrate an exceptional example. Assume the execution time of task t_1 is longer than data transmission time of d_1. As shown by subgraph (3), if the coordinator peer starts staging d_1 too early, the virtual machine needs to wait for the output data of t_1 which will lead to unnecessary cost.

In this paper, we enhance our previous algorithm by utilizing data staging to overlap task execution and data transmission. While a task starts execution, for example task t_1 in Figure 6, the coordinator peer checks whether it is possible to stage data for its subsequent task, that is, t_2. This is done by checking if other input data of the child is ready. If true, the coordinator peer will plan for staging. As mentioned previously, the time for starting should be carefully estimated to facilitate data staging. Firstly, the end time of the task is estimated: $t_{Endj} = t_{Startj} + t_{Ej} + t_{Gj}$. Then, the transmission time of all other input data is calculated as t_{Di} or t_{Gi} which is for transferring dataset d_i or generated dataset of task t_i, respectively. Finally, the time for starting data staging is set by $\max(t_{Startj}, t_{Endj} - t_{Di}, t_{Endj} - t_{Di})$. In this way, even though we can only use estimated time, the overlapped time between task execution and data transmission is masked to some extent. Furthermore, the idle waiting time for long running tasks can also be reduced.

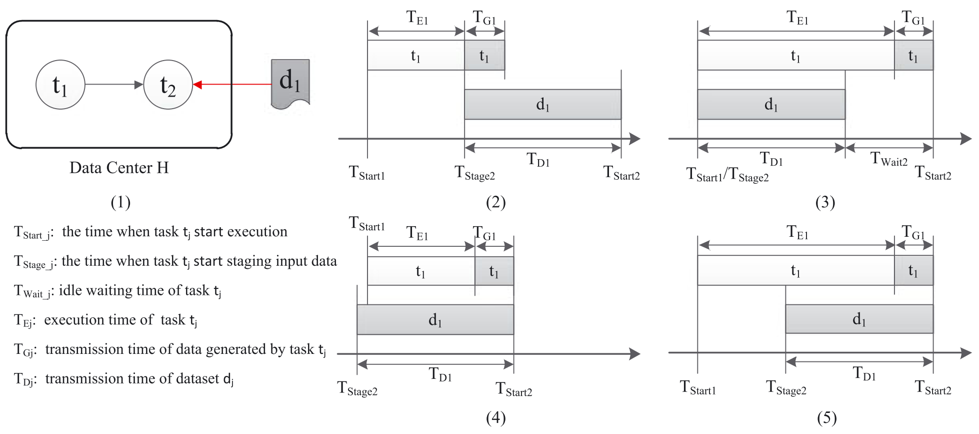

t1
t2
d1
Data Center H
(1)
TE1
TG1
TD1
TStart1
TStage2
TStart2
(2)
TWait2
TStart1/TStage2
(3)
(4)
(5)
TStart_j: the time when task tj start execution
TStage_j: the time when task tj start staging input data
TWait_j: idle waiting time of task tj
TEj: execution time of task tj
TGj: transmission time of data generated by task tj
TDj: transmission time of dataset dj

Figure 6. Examples of data staging.

5. PERFORMANCE EVALUATION

5.1. *Simulation settings*

The evaluation of our strategy was performed on a simulation platform built on the Tianhe cluster which is a homogeneous product but with a smaller scale of the Tianhe-1A supercomputer that ranked first in Top500 in June 2010. The simulation environment consists of 40 computing nodes on the Tianhe cluster. Each node contains two Intel Xeon E5540 2.53GHz quad-core CPUs running Redhat Linux AS 5.3. To simulate cloud computing environment, Xen [12] was installed on every node, and virtual clusters were created as data centers. We deployed the Apache HDFS on every data center for managing distributed application data and ran SwinDeW-C [14] upon the data centers to interpret and execute workflows. The bandwidth between the data centers is limited to the range of [2MBps, 10MBps].

To demonstrate the effectiveness of the strategy, we compare it with the previous *k*-means clustering based data placement strategy (Cluster-D) and a random strategy. We refer to our strategy as Cluster-C (Clustering based Coscheduling) and Cluster-S (Clustering with Staging). These strategies are briefly described in the following:

- *Random*: In this strategy, input datasets are randomly placed in the data centers at the very start. The intermediate datasets are stored in the local data center whenever it has sufficient free space. Otherwise, they are placed in randomly selected data centers that can accommodate them. This strategy represents the conventional data placement algorithm used in early grid and/or cluster systems where datasets are naturally stored locally or placed in randomly selected nodes that have available storage.
- *Cluster-D*: This strategy is the one proposed in [6], designed for executing data-intensive scientific workflows in collaborative cloud environments. The basic idea is to cluster the input datasets into *k* partitions and distribute each partition into an appropriate data center at build-time stage. When a new dataset is generated at runtime stage, it will be distributed into the data center that has the highest dependency with it.
- *Cluster-C*: It represents our prior weighted *k*-means clustering based coscheduling strategy. The differences between Cluster-C and Cluster-D are threefold. Firstly, Cluster-C takes the data size into consideration and utilizes dependencies among datasets and tasks. Secondly, it implements a node adjustment algorithm to deal with occasionally bad distributions. Finally, task duplication is used in Cluster-C to trade computation cost for communication cost.
- *Cluster-S*: Similar to Cluster-C, this strategy clusters the input datasets and tasks into *k* partitions in such a way that each partition gets the same amount of input data. However, unlike Cluster-C that starts a task when all the input datasets are ready, Cluster-S starts a task even while its preceding tasks are generating its input data. This is done by staging data that are already generated in parallel with task execution. Thus, data transmission and task execution is overlapped, and the makespan is decreased. However, this technique cannot reduce the amount of data transfer during workflow execution.

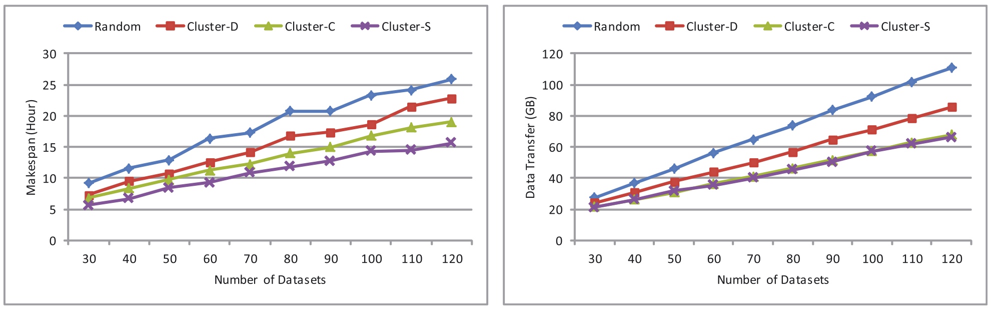
Generally, reducing the number of data movements can reduce the execution time and the volume of data transfer as discussed in our previous work [6]. However, different datasets may have different sizes and it may be more efficient to move several small datasets than moving a large one. Therefore, in this paper, we improved our previous strategy and demonstrate the effectiveness of our strategy in terms of total volume of data transfer among distributed data centers. Besides, we also use workflow makespan to assess the performance of the different strategies. The makespan of a workflow is calculated by the time elapsed from the start of its first task until the completion of the last task.

5.2. *Simulation of general workflows and results*

In the following experiments, we first consider randomly generated scientific workflows. A random workflow graph generator was implemented to generate DAGs with various characteristics specified by several input parameters, including (i) number of tasks in the graph; (ii) number of input datasets; (ii) range of execution time for the tasks; (iv) range of data size for input datasets and output datasets; and (v) maximum out-degree for datasets and tasks in the graph. In our random simulations, we generated extensive workflow graphs for evaluation. Without losing generality, we present the representative results in this section. Particularly, we set the number of tasks equal to the number of datasets. The sizes of input datasets are set by the range [1MB, 1024 MB] . The maximum usage of a dataset is set by 3 and the out-degree (maximum usage) of a task is set by 2. The execution time of a task is randomly set between 1 second and 15 minutes. To prevent biasing to a particular strategy, we ran 1000 test workflows under the same configuration and took the average value as the final result.

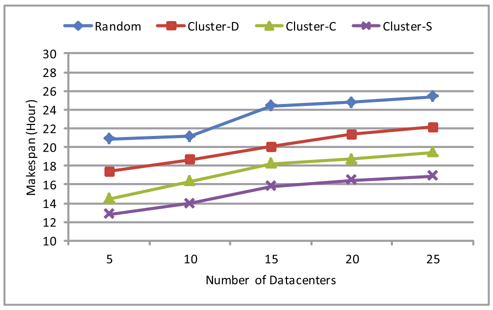
In the first evaluation, we ran test workflows with different numbers of datasets on 10 data centers. Figure 7 shows the increment of the execution time as well as the total volume of data transfer when the number of datasets in the workflows increases. From the graphs, we can see that the coscheduling strategy outperforms the none-coscheduling strategies in all cases. In fact, Cluster-C reduces the makespan of the test workflows averagely by 27.90% and 12.60% in comparison with Random strategy and Cluster-D strategy, respectively. The total volume of data transfer introduced by Cluster-C is reduced by 34.60% and 17.79% on average in comparison with Random and Cluster-D strategies. We attribute the efficiency of the Cluster-C strategy to the following advancements: (i) in addition to data–data dependency, the strategy defines and utilizes data–task and task–task dependency, which can group the most related tasks and datasets together; (ii) Cluster-C also takes data size into consideration when calculating the weight of dependencies, as such it transfers less volume of datasets in comparison with Cluster-D strategy; (iii) in Cluster-C, we utilize a node adjustment algorithm and the task duplication technique to rectify possible bad placement. However, we can see from the figures that the disparity between Cluster-D and Cluster-C in Figure 7(a) is a little less than that in Figure 7(b). This is because Cluster-C duplicates tasks which introduce additional execution time. By the enhancement of data staging, the makespan in Cluster-C can further be reduced. Actually, the makespan in Cluster-S is averagely decreased by 16.14% in comparison with Cluster-C, with a minimum reduction of 11.33% for 70 datasets and a maximum reduction of 19.97% for 110 datasets. However, the amount of data transfer in Cluster-S is basically the same as that in Cluster-C because data staging cannot reduce the amount of data transfer.

Figure 8 shows the results of running the workflows in different numbers of data centers. In this test, the number of datasets and the number of tasks are both set by 100. Like the first simulation, we evaluate the makespan and total volume of data transfer while the size of output data is set by the range [1MB, 1024 MB]. From Figures 8(a) and (b), we can see that both the makespan and data transfer in Random strategy first greatly increases until the number of data centers is 15, then the rate decreases. The reason is that the number of input datasets in the simulation is 100, which means that the average number of datasets in each data center is 6, 5, and 4, while the number of data centers is 15, 20, and 25, respectively. Therefore, the values of the two metrics just slightly increase because the average number of input datasets in these cases is similar. However, in Cluster-D and Cluster-C strategies, the values continue to increase with a greater rate. This is because both strategies are more sensitive to the storage constraints introduced by the load balance of input datasets. Nonetheless, in comparison with Random strategy and Cluster-D strategy, the Cluster-C strategy reduces the makespan of the test

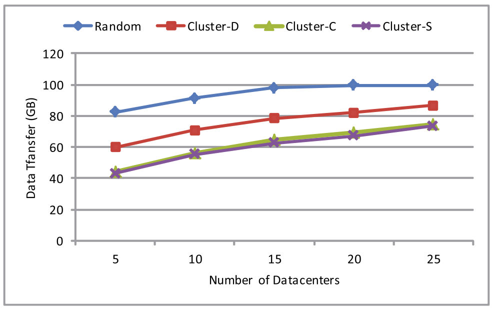


(a) Makespan. (b) Data Transfer.

Figure 7. Random simulation with different numbers of datasets.



(a) Makespan.



(b) Data Transfer.

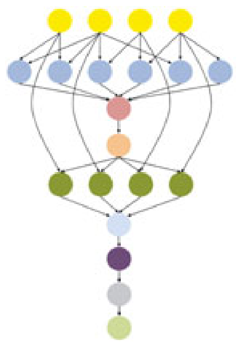
Figure 8. Random simulation with different numbers of data centers.

workflows by 25.28% and 12.61% and reduces the total volume of data transfer by 34.99% and 18.98%, respectively. Through data staging, the makespan in Cluster-S simulation is further declined by 12.89% averagely. Particularly, because the number of datasets in a data center declines adversely with the number of data centers, the reduction of execution time achieves its maximum (14.76%) when the number of data centers is 10. Later, the reduction keeps steady as the number of datasets and tasks in each data center changes very slightly.

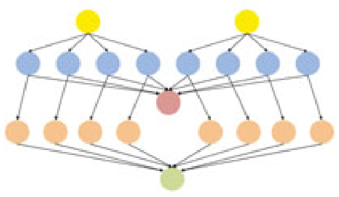
5.3. *Simulation of real workflows and results*

The random simulations illustrate the general performance achieved by our strategy. In this subsection, realistic workflows are used to show the effectiveness of the proposed strategy in real world scientific applications. The workflows for testing are Montage and CyberShake, respectively. Their approximate structures are shown in Figure 9. We can see that Montage gets more types of tasks and has a deeper pipeline than CyberShake. Even though the CyberShake workflow only includes five types of tasks and an extremely short execution flow, it requires large numbers of data dissemination and aggregation. The description of these scientific workflows and the execution results are presented as follows.

The Montage workflow is a data-intensive scientific workflow in which the input datasets and the intermediate data that generated during execution are of considerable size [15]. However, the tasks have a small runtime which is approximately several minutes. The detailed information about Montage is publicly available at [16]. In this simulation, we utilize the Montage workflow with a size of 1000 tasks for evaluation. The execution profile of Montage_1000 is shown in Table II. Moreover, we ran the test workflow under different numbers of data centers and the result is presented in Figure 10.



(a) Montage.

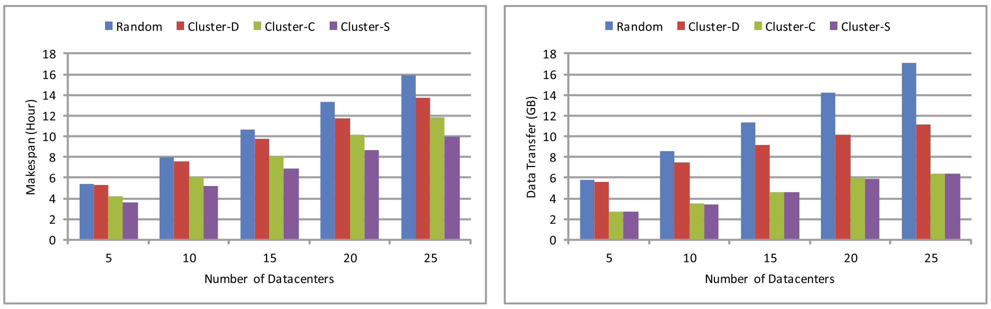


(b) CyberShake.

Figure 9. Structures of two data-intensive realistic scientific workflows.

Table II. Execution profile of montage workflow.

Task	Count	Average runtime (s)	Average input size (MB)	Average output size (MB)
mBgModel	1	89.12	0.16	0.01
mImgTbl	1	65.91	1317.53	0.06
mShrink	1	22.25	581.30	11.63
mConcatFit	1	52.96	0.19	0.13
mDiffFit	662	10.59	15.85	2.67
mProjectPP	166	13.58	4.03	7.94
mJPEG	1	2.52	11.63	1.36
mAdd	1	99.53	0.06	581.30
mBackground	166	10.74	7.95	7.94



(a) Makespan. (b) Data Transfer.

Figure 10. Montage workflow with different numbers of data centers.

From the figures, we can see that both the makespan and volume of data transfer increase along with the increment of data centers. However, the volume of data transfer in both Cluster-D and Cluster-C increases much slower than that in Random strategy. It is noticeable to see that the difference between Cluster-C and Cluster-D in makespan is less than that in data transfer. The reason is that the Cluster-C strategy duplicates tasks to trade computation cost for communication cost, which reduces the total volume of data transfer but introduces additional computation time. In comparison with Random and Cluster-D strategies, Cluster-C reduces workflow makespan by 23.87% and 17.13% and total volume of data transfer by 58.71% and 48.04%, respectively. This result shows that the coscheduling strategy can be more advantageous in real-world applications. Moreover, when the input datasets are staging beforehand, the makespan for the Montage workflow can be further reduced by 14.45% averagely in comparison with the coscheduling-only strategy.

In the second simulation, we utilize the CyberShake workflow with 1000 tasks for evaluation. The CyberShake workflow is both data-intensive and computational intensive and is used by seismologists and engineers to characterize earthquake hazards in a region through the probabilistic seismic hazard analysis technique [17]. The execution profile of CyberShake_1000 is shown in Table III. Because the number of input for ExtractSGT is small but the volume is excessively large, we split the input

Table III. Execution profile of CyberShake workflow.

Task	Count	Average runtime (s)	Average input size (MB)	Average output size (MB)
ZipSeis	1	9.38	11.38	2.04
PeakValCalcOkaya	497	1.08	0.023	0.0002
SeismogramSynthesis	497	43.65	513.74	0.023
ExtractSGT	4	126.12	38148.33	510.34
ZipPSA	1	3.81	0.10	0.01

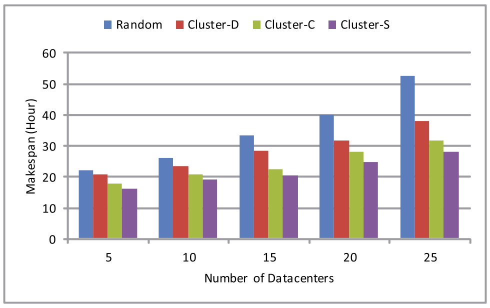
data into 100 pieces. Therefore, when ExtractSGT is started, it needs to retrieve the pieces and assemble them into an intact component. Figure 11 illustrates the result of executing the CyberShake workflow in different numbers of data centers.

The makespan of CyberShake workflow is consistent with that of Montage workflow. In comparison with Random and Cluster-D, Cluster-C reduces the execution time by 28.55% and 15.01%. Moreover, through data staging, the makespan is further reduced by 9.98% compared with the Cluster-C strategy. This means that the effect of data staging is not significant for CyberShake. The reason is that the most of the data transmission and computation take place in ExtractSGT and SeismogramSynthesis which are the first two steps in CyberShake. Hence, data staging can only play its role while ZipSeis retrieves input data from SeismogramSynthesis tasks. For data transfer, Cluster-D achieves a 36.35% reduction in comparison with Random strategy. However, the coscheduling strategies display a much smaller reduction compared with Cluster-D strategy, with a maximum of 7.03%. This is because in CyberShake workflow, the data transfer is centralized in the first two steps, and the scheduling strategy in Cluster-D, which schedules the task to the data center that has the most input datasets, is effective for CyberShake. The small reduction in Cluster-C is mainly due to the adjustment algorithm.

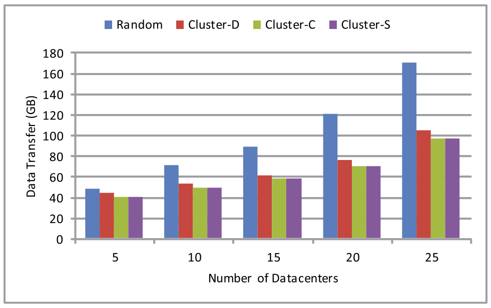
6. RELATED WORK

Because workflow is an important abstraction for performing scientific experiments in computational environments, the problem of scheduling scientific workflows has been intensively studied in both traditional grid computing and current cloud computing environments. For example, Rahman *et al.* [18] presented a reputation-based grid workflow scheduling algorithm which can dynamically adapt to changing resource conditions and achieve good performance. In [19], a QoS-based scheduling algorithm using the partial critical paths is proposed with the goal of minimizing the cost of workflow execution while fulfilling Service Level Agreements (SLAs) in utility grids. Bessai *et al.* [20] presented several resource allocation and scheduling methods to make tradeoff between cost and makespan of running scientific workflow applications in the clouds. Ostermann and Prodan [21] studied using spot instances in commercial clouds to extend grid resources for large-scale scientific workflow execution. Besides cost and makespan, Oliveira *et al.* [22] considered reliability as the third criteria and provided methods for dynamical resource adjustment based on scientific workflow provenance data. However, none of the aforementioned work addresses the issue of data placement which may cause large latency while performing data-intensive scientific workflows in collaborative cloud environments.

For managing extremely large datasets and to deliver high throughput for large-scale data-intensive applications, new data management systems have emerged along with cloud computing, such as the Google File System [23] and the Hadoop Distributed File System [13]. However, they are designed for one data center, and the data management for multiple data centers remains a big challenge.



(a) Makespan.



(b) Data Transfer.

Figure 11. CyberShake workflow with different numbers of data centers.

Sector [24] is a storage cloud that supports not only data storage for one data center but also data distribution across WANs by using UDP for message passing and UDT [25] for data transfer. In workflow systems, existing toolkits are utilized for data management. For instance, Kepler [26] uses the SDSC Storage Resource Broker [27] to obtain and put files, Pegasus [28] and TRIANA [29] use the Globus Replica Location Service [30] to locate required datasets and utilize GridFTP [31] for data transmission. Nonetheless, these systems try to minimize data transfer latency by providing fast data transfer.

Stork [32] data placement scheduler provides solutions for efficient and reliable data placement over WANs by queuing, scheduling, and even checkpointing the data placement activities. Adaptive Overlapped Declustering [33] is a placement method designed for parallel storage systems to improve space utilization by balancing their access loads. Perfect Balancing (PB) [34] is a static data placement strategy for distributed storage clusters to reduce response time through load-balancing. In [35], the authors proposed a data placement algorithm which can provide grid users with automatic and intelligent data placement by considering the characteristics of the datasets such as their sizes and popularities. In [36], Cope *et al.* proposed a series of robust data placement techniques to perform time critical workflows in heterogeneous urgent computing systems. However, these approaches seem to be at the opposite extreme because they only use simple heuristics for task scheduling. Therefore, based on the previous work [6], a coscheduling strategy was proposed to couple data placement with task scheduling for efficient execution of scientific workflows in cloud environments [7].

Data staging is widely studied in grid computing and high performance computing environments to improve application performance. Chervenak *et al.* [37] studied the impact of data placement services on scientific workflow management systems in which they found that data prestaging can significantly improve application performance. In [38], a grid scheduling system was proposed for tightly coupling data management with task scheduling. Particularly, data staging is leveraged to overlap task execution to gain high CPU utilization. Even though data staging can reduce the waiting time for task execution, staging data far in advance is not always good because it wastes expensive scratch storage [39]. To avoid the shortage, Monti *et al.* [40] proposed the just-in-time staging framework for staging input data from end-user location into the scratch parallel file system of a supercomputer just before an application is about to run. Inspired by previous work, we used data staging to speed up the start time of the tasks. Although the volume of data transfer cannot be reduced, the makespan of the tested scientific workflows can be decreased noticeably.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a coscheduling strategy based on our previous studies for efficient execution of scientific workflows in collaborative cloud environments that consist of a number of data centers. In detail, we defined three types of dependencies among datasets and tasks and extended our previous *k*-means data placement strategy with task scheduling for the goal of distributing input datasets evenly into multiple data centers while the most related datasets and tasks are placed together. Furthermore, we presented an adjustment algorithm to correct possibly bad grouping and utilized the task duplication technique to trade computation cost for communication cost. The proposed algorithms have been implemented in the SwinDeW-C workflow management system for build-time planning. To start tasks earlier during workflow execution, data staging is used to retrieve input datasets in advance to overlap task computation with data transmission. Simulation results through random and real world workflows demonstrated that our strategy outperforms other representative strategies in terms of makespan and total volume of data transfer.

Future studies in this research can be performed in the following directions. First, we are exploring data replication strategies to trade storage cost with communication cost. Thus, additional algorithms will be developed to determine which dataset is the best candidate to be replicated and where it should be placed. Second, in this paper, we have evaluated our strategy mainly in terms of makespan and the total volume of data transfer. However, one of the most attractive feature of cloud computing is its cost effectiveness. Therefore, in the future, we plan to evaluate the proposed strategy in popular cloud platforms such as Amazon EC2 and refer to the regular pricing models to show the cost effectiveness of our strategy. Although in this paper we used two data-intensive realistic scientific workflows with very different structures for evaluation, there is still a lack of realistic data-intensive workflows with characterized structures. In the future, we will use more types of scientific workflows both in the form of structures and resource requirements for evaluation.

REFERENCES

1. Kaijun R, Nong X, Jinjun C. Building quick service query list using WordNet and multiple heterogeneous ontologies toward more realistic service composition. *IEEE Transactions on Services Computing* 2011; **4**(3): 216–229.
2. Keahey K, Figueiredo R, Fortes J, *et al*. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. Cloud Computing and Its Applications, Chicago, IL, 2008.
3. Christina H, Gaurang M, Timothy F, *et al*. On the Use of Cloud Computing for Scientific Workflows. *IEEE Fourth International Conference on eScience*, 2008; 640–645.
4. Ewa D, Gurmeet S, Miron L, *et al*. The Cost of Doing Science on the Cloud: The Montage Example. *ACM/IEEE conference on Supercomputing*, Austin, Texas, 2008; 1–12.
5. Iosup A, Ostermann S, Yigitbasi MN, *et al*. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems* 2011; **22**(6):931–945.
6. Dong Y, Yun Y, Xiao L, *et al*. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems* 2010; **26**(8):1200–1214.
7. Kefeng D, Lingmei K, Junqiang S, *et al*. A Weighted K-Means Clustering Based Co-Scheduling Strategy Towards Efficient Execution of Scientific Workflows in Collaborative Cloud Environments. *IEEE Ninth International Conference on Dependable*, *Autonomic and Secure Computing* (*DASC*), 2011; 547–554.
8. Arabie P, Hubert LJ. The bond energy algorithm revisited. *IEEE Transactions on Systems, Man, and Cybernetics* 1990; **20**(1):268–274.
9. Sinnen O, To A, Kaur M. Contention-aware scheduling with task duplication. *Journal of Parallel and Distributed Computing* 2011; **71**(1):77–86.
10. Tang X, Li K, Liao G, *et al*. List scheduling with duplication for heterogeneous computing systems. *Journal of Parallel and Distributed Computing* 2010; **70**(4):323–329.
11. Yun Y, Ke L, Jinjun C, *et al*. Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G. *IEEE International Conference on e-Science and Grid Computing*, 2007; 51–58.
12. Barham P, Dragovic B, Fraser K, *et al*. Xen and the Art of Virtualization. *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, New York, NY, USA, 2003; 164–177.
13. Le K, Bianchini R, Zhang J, *et al*. Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds. *Proceedings of 2011 International Conference for High Performance Computing*, *Networking*, *Storage and Analysis*, Seattle, Washington, 2011; 1–12.
14. Xiao L, Dong Y, Gaofeng Z, *et al*. SwinDeW-C: A Peer-to-Peer Based Cloud Workflow System for Managing Instance Intensive Applications. Handbook of Cloud Computing. Springer, 2010.
15. Shishir B, Ann C, Ewa D, *et al*. Characterization of Scientific Workflows. *Workshop on Workflows in Support of Large Scale Science*, 2008.
16. WorkflowGenerator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator [7.10.2012]
17. Deelman E, Callaghan S, Field E, *et al*. Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example. *Second IEEE International Conference on e-Science and Grid Computing*, 2006; 14–14.
18. Rahman M, Ranjan R, Buyya R. Dependable Workflow Scheduling in Global Grids. *10th IEEE/ACM International Conference on Grid Computing*, 2009; 153–162.
19. Abrishami S, Naghibzadeh M, Epema D. Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths. *11th IEEE/ACM International Conference on Grid Computing* (*GRID*), 2010; 81–88.
20. Bessai K, Youcef S, Oulamara A, *et al*. Bi-Criteria Workflow Tasks Allocation and Scheduling in Cloud Computing Environments. *IEEE 5th International Conference on Cloud Computing* (*CLOUD*), 2012; 638–645.
21. Ostermann S, Prodan R. Impact of variable priced cloud resources on scientific workflow scheduling. Euro-Par 2012 Parallel Processing (Lecture Notes in Computer Science Vol. 7484). Springer Berlin Heidelberg, 2012; 350–362.
22. Oliveira D, Ocaña KCS, Baião F, *et al*. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *Journal of Grid Computing* 2012; **10**(3):521–552.
23. Sanjay G, Howard G, Shun-Tak L. The Google file system. Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 2003; 29–43.
24. Gu Y, Grossman RL. Sector and sphere: the design and implementation of a high-performance data cloud. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 2009; **367**(1897):2429.
25. Yunhong G, Robert LG. UDT: UDP-based data transfer for high-speed wide area networks. *Computer Networks (Elsevier)* 2007; **51**(7).
26. Lud scher B, Altintas I, Berkley C, *et al*. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1039–1065.

27. Chaitanya B, Reagan M, Arcot R, *et al*. The SDSC Storage Resource Broker. *IBM Centre for Advanced Studies Conference*, Toronto, Ontario, Canada, 1998; 1–12.
28. Deelman E, Blythe J, Gil Y, *et al*. Pegasus: Mapping Scientific Workflows Onto the Grid. *European Across Grids Conference*, 2004; 11–20.
29. Churches D, Gombas G, Harrison A, *et al*. Programming scientific and distributed workflow with Triana services. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1021–1037.
30. Chervenak A, Deelman E, Foster I, *et al*. Giggle: a framework for constructing scalable replica location services. ACM/IEEE Conference on Supercomputing, Baltimore, Maryland, 2002; 1–17.
31. Allcock B, Bester J, Bresnahan J, *et al*. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. *Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, 2001; 13–13.
32. Tevfik K, Miron L. Stork: Making Data Placement a First Class Citizen in the Grid. *24th International Conference on Distributed Computing Systems*, 2004; 342–349.
33. Watanabe A, Yokota H. Adaptive Overlapped Declustering: A Highly Available Data-Placement Method Balancing Access Load and Space Utilization. *21st International Conference on Data Engineering*, 2005; 828–839.
34. Madathil DK, Thota RB, Paul P, *et al*. A Static Data Placement Strategy Towards Perfect Load-Balancing for Distributed Storage Clusters. *IEEE International Parallel and Distributed Processing Symposium*, 2008; 1–8.
35. Ying D, Ying L. Automatic Data Placement and Replication in Grids. International Conference on High Performance Computing, 2009; 30–39.
36. Cope JM, Trebon N, Tufo HM, *et al*. Robust Data Placement in Urgent Computing Environments. *IEEE International Symposium on Parallel & Distributed Processing*, 2009; 1–13.
37. Chervenak A, Deelman E, Livny M, *et al*. Data Placement for Scientific Applications in Distributed Environments. *8th IEEE/ACM International Conference on Grid Computing*, 2007; 267–274.
38. Machida Y, Takizawa Si, nakada H, *et al*. Intelligent data staging with overlapped execution of grid applications. *Future Generation Computer Systems* 2008; **24**(5):425–433.
39. Zhang Z, Wang C, Vazhkudai SS, *et al*. Optimizing Center Performance Through Coordinated Data Staging, Scheduling and Recovery. *ACM/IEEE Conference on Supercomputing*, 2007; 1–11.
40. Monti HM, Butt AR, Vazhkudai SS. Just-in-Time Staging of Large Input Data for Supercomputing Jobs. *3rd Petascale Data Storage Workshop* (*PDSW*), 2008; 1–5.