A data placement strategy in scientific cloud workflows

Dong Yuan, Yun Yang, Xiao Liu, Jinjun Chen

PII: S0167-739X(10)00020-8
DOI: 10.1016/j.future.2010.02.004
Reference: FUTURE 1843

To appear in: *Future Generation Computer Systems*

Received date: 30 May 2009
Revised date: 26 November 2009
Accepted date: 1 February 2010

A Data Placement Strategy in Scientific Cloud Workflows

Dong Yuan, Yun Yang, Xiao Liu, Jinjun Chen
Faculty of Information and Communication Technologies,
Swinburne University of Technology
Hawthorn, Melbourne, Australia 3122
{dyuan, yyang, xliu, jchen}@swin.edu.au

ABSTRACT

In scientific cloud workflows, large amounts of application data need to be stored in distributed data centres. To effectively store these data, a data manager must intelligently select data centres in which these data will reside. This is, however, not the case for data which must have a fixed location. When one task needs several datasets located in different data centres, the movement of large volumes of data becomes a challenge. In this paper, we propose a matrix based *k*-means clustering strategy for data placement in scientific cloud workflows. The strategy contains two algorithms that group the existing datasets in *k* data centres during the workflow build-time stage, and dynamically clusters newly generated datasets to the most appropriate data centres - based on dependencies - during the runtime stage. Simulations show that our algorithm can effectively reduce data movement during workflow execution.

Keywords-data management; scientific workflow; cloud computing;

1. INTRODUCTION

Running scientific workflow applications usually need not only high performance computing resources but also massive storage [18]. In many scientific research fields, like astronomy [17], high-energy physics [35] and bio-informatics [39], scientists need to analyse terabytes of data either from existing data resources or collected from physical devices. During these processes, similar amounts of new data might also be generated as intermediate or final products [18]. Workflow technologies are facilitated to automate these scientific applications. Scientific workflows are typically very complex. They usually have a large number of tasks and need a long time for execution. Nowadays, popular scientific workflows are deployed in grid systems [35] because they have high performance and massive storage. However, building a grid system is extremely expensive and it is not available for scientists all over the world to use.

The emergence of cloud computing technologies offers a new way to develop scientific workflow systems. Since late 2007 the concept of cloud computing was proposed [47] and it has been utilised in many areas with some success [8] [25] [10] [38]. Cloud computing is deemed as the next generation of IT platforms that can deliver computing as a kind of utility [11]. Foster et al. made a comprehensive comparison of grid computing and cloud computing [23]. Some features of cloud computing also meet the requirements of scientific workflow systems. First, cloud computing systems can provide high performance and massive storage required for scientific applications in the same way as grid systems, but with a lower infrastructure construction cost among many other features, because cloud computing systems are composed of data centres which can be clusters of commodity hardware. Second, cloud computing systems offer a new paradigm that scientists from all over the world can collaborate and conduct their research together. Cloud computing systems are based on the Internet, and so are the scientific workflow systems deployed on the cloud. Dispersed computing facilities (like clusters) at different institutions can be viewed as data centres in the cloud computing platform. Scientists can upload their data and launch their applications on scientific cloud workflow systems from anywhere in the world via the Internet. As all the data are managed on the cloud, it is easy to share data among scientists. Research into doing science on the cloud has already commenced such as early experiences like Nimbus [31] and Cumulus [46] projects. The work by Deelman et al. [20] shows that cloud computing offers a cost-effective solution for data-intensive applications, such as scientific workflows [29].

By taking advantage of cloud computing, scientific workflow systems could gain a wider utilisation; however they will also face some new challenges, where data management is one of them. Scientific applications are data intensive and usually need collaborations of scientists from different institutions [6], hence application data in scientific workflows are usually distributed and very large. When one task needs to process data from different data centres, moving data becomes a challenge [18]. Some application data are too large to be moved efficiently, some may have fixed locations that are not feasible to be moved and some may have to be located at fixed data centres for processing, but these are only one aspect of this challenge. For the application data that are flexible to be moved, we also cannot move them whenever and wherever we want, since in the cloud computing platform, data centres may belong to different cloud service providers that data movement would result in costs. Furthermore, the infrastructure of cloud computing systems is hidden from their users. They just offer the computation and storage resources required by users for their applications. The users do not know the exact physical locations where their data are stored. This kind of model is very convenient for users, but remains a big challenge for data management to scientific cloud workflow systems.

In this paper, we propose a matrix based *k*-means clustering strategy for data placement in scientific cloud workflow systems. Scientific workflows can be very complex, one task might require many datasets for execution; furthermore, one dataset might also be required by many tasks. If some datasets are always used together by many tasks, we say that these datasets are dependant on each other. In our strategy, we try to keep these datasets in one data centre, so that when tasks were scheduled to this data centre, most, if not all, of the data they need are stored locally.

Our data placement strategy has two algorithms, one for the build-time stage and one for the runtime stage of scientific workflows. In the build-time stage algorithm, we construct a dependency matrix for all the application data, which represents the dependencies between all the datasets including the datasets that may have fixed locations. Then we use the BEA algorithm [37] to cluster the matrix and partition it that datasets in every partition are highly dependent upon each other. We distribute the partitions into *k* data centres, where the partitions have fixed location datasets are also placed in the appropriate data centres. These *k* data centres are initially as the partitions of the *k*-means algorithm at runtime stage. At runtime, our clustering algorithm deals with the newly generated data that will be needed by other tasks. For every newly generated dataset, we calculate its dependencies with all *k* data centres, and move the data to the data centre that has the highest dependency with it.

By placing data with their dependencies, our strategy attempts to minimise the total data movement during the execution of workflows. Furthermore, with the pre-allocate of data to other data centres, our strategy can prevent data gathering to one data centre and reduces the time spent waiting for data by ensuring that relevant data are stored locally.

The remainder of the paper is organised as follows. Section 2 presents the related work. Section 3 gives an example and analyses the research problems. Section 4 introduces the basic strategy of our algorithms. Section 5 presents the detailed steps of the algorithms in our data placement strategy. Section 6 demonstrates the simulation results and the evaluation. Finally, Section 7 addresses our conclusions and future work.

2. Related work

Data placement of scientific workflows is a very important and challenging issue. In traditional distributed computing systems, much work about data placement has been conducted. In [49], Xie proposed an energy-aware strategy for data placement in RAID-structured storage systems. Stork [33] is a scheduler in the Grid that guarantees that data placement activities can be queued, scheduled, monitored and managed in a fault tolerant manner. In [15], Cope et al. proposed a data placement strategy for urgent computing environments to guarantee data robustness. At the infrastructure level, NUCA [28] is a data placement and replication strategy for distributed caches that can reduce data access latency. However, none of them focuses on reducing data movement between data centres on the Internet. As cloud computing has become more and more popular, new data management systems have also appeared, such as Google File System [24] and Hadoop [3]. They all have hidden infrastructures that can store the application data independent of users' control. Google File System is designed mainly for Web search applications, which are different from workflow applications. Hadoop is a more general distributed file system, which has been used by many companies, such as Amazon and Facebook. When you push a file to a Hadoop File System, it will automatically split this file into chunks and randomly distribute these chunks in a cluster. Furthermore, the Cumulus project [46] introduced a scientific cloud architecture for a data centre. And the Nimbus [31] toolkit can directly turn a cluster into a cloud and it has already been used to build a cloud for scientific applications. Within a small cluster, data movement is not a big problem, because there are fast connections between nodes, i.e. Ethernet. However, the scientific cloud workflow system is designed for scientists to collaborate, where large scale and distributed applications need to be executed across several data centres. The data movement between data centres may cost a lot of time, since data centres are spread around the Internet with limited bandwidth. In this work, we try to place the application data based on their dependencies in order to reduce the data movement between data centres.

Data transfer is a big overhead for scientific workflows [41]. Though popular scientific workflow systems have their data management strategies, they did not focus on reducing data movement. For the build-time stage, these systems mainly focus on data modelling methods. For example, Kepler [35] has an actor-oriented data modelling method that works for large data in a grid environment, Taverna [39] and ASKALON [48] have their own process definition language to represent their data flows. For the runtime stage, most of the scientific workflow systems adopt some data grid systems for their data management. For examples, Kepler uses the SRB [7] system, while Pegasus [17] and Triana [14] adopt the RLS system [12], Gridbus [9] has a grid service broker [45] where all data are deemed as important resources. Data grids primarily deal with providing services and infrastructure for distributed data-intensive applications that need to access, transfer, and modify massive datasets stored in distributed storage resources [44]. However, these systems do not consider the dependencies between data in scientific workflows either at build-time or runtime and they also can not reduce data movement. Some researches in grid computing have addressed the importance of data dependency for the large-scale scientific applications, although they did not focus on workflow data management. The Filecules project [21] groups the files based on the dependencies. Using real workload experiments data, the authors demonstrated that filecules grouping is a reliable and useful abstraction for data management in science Grid. BitDew [22] is a distributed data management system for desktop Grid. Different from data centres in the cloud that aim to provide services to users, desktop Grid aims to make use of the idle computing and storage resources in the desktop computers. In BitDew, the data placement dependency is denoted by a data attribute called "affinity", which is pre-defined by users. However, in cloud computing, all the applciation data are hosted in the data centres, where anyone can use the cloud services and upload their data. Letting users define the data dependencies for the scientific cloud workflows is clearly impractical.

The closest workflow research to ours is the Pegasus workflow system which has proposed some data placement strategies [13] [42] based on the RLS system. The strategies are: first, pre-allocate the required data to the computation resource where the task will execute; second, dynamically delete the data that will no longer be used by tasks. These strategies are only for the runtime stage of scientific workflows and can effectively reduce the overall execution time and the storage usage of the workflows. Furthermore, in [32], the authors proposed a data placement scheduler for distributed computing systems. It guarantees the reliable and efficient data transfer with different protocols. These works mainly focus on how to move the application data, and they can not reduce the total data movement of the whole system. However, our work aims to reduce data movement. Our strategy is for both build-time and runtime stages of scientific workflows and we design specific algorithms to automatically place and move the application data.

In cloud computing systems, the infrastructure is hidden from users. Hence, for most of the application data, the system will decide where to store them. Dependencies exist among these data. In this paper, we initially adapt the clustering algorithms for data movement based on data dependency. Clustering algorithms have been used in pattern recognition since 1980s [30], which can classify patterns into groups without supervision. Today they are widely used to process data streams [27]. In many scientific workflow applications, the intermediate data movement is in data stream format and the newly generated data must be moved to the destination in real-time. We adapt the *k*-means clustering algorithm for data placement. When new data is generated by a task, we dynamically calculate the dependencies of the new data with the *K* data centres, and move the new data to the centre with highest dependency. The simulation results of this paper show that with our data placement strategy, the data movement between data centres is significantly reduced compared to random data placement.

3. Scientific cloud workflow Data Management

3.1. *A Motivating Example*

Scientific applications often need to process terabytes of data. For example, the ATNF[1] Parkes Swinburne Recorder (APSR) [2] is a next-generation baseband data recording and processing system currently under development in collaboration by Swinburne University of Technology and ATNF. The data from the APSR streams at a rate of one gigabyte per second. The researchers at Parkes process the data with a local cluster of servers and do their research. All the data are stored locally at Parkes and they are not available to other institutions. If researchers at other institutions need the data resources from the Parkes Radio Telescope, they have to contact the researchers at Parks and request for the data. Researchers at Parkes will check the local repositories to see if the existing data resources could fulfill the requirements. In this situation communications often suffer from low efficiency because researchers are from different projects and the requirements are usually complex. Sometimes researchers even have to go to Parkes and bring back the data that they need on hard disks. Sharing data resources in this manner is obviously inefficient and hence not desirable.

With cloud computing technologies, we can turn the Parkes cluster into a data centre on a cloud computing platform that can offer services to researchers all over the world. The cloud computing platform is built on the Internet, which is how the data centres are connected to each other. All the data are managed by the cloud data management system. The researchers can access the existing data resources, upload application data and launch their applications via the cloud service. By doing this, the

[1] ATNF refers to the Australian Telescope National Facility.

resources at Parkes will be fully utilised, since data can be sent to other data centres for different applications as needed. On the other hand, researchers at Parkes will be able to do more scientific research by retrieving useful data from other data centres around the world. All these data sending and retrieving operations are hidden from the researches. In another word, via cloud computing platform, researchers can utilise data resources from other institutions without knowing where the data are physically stored. Hence, on a cloud computing platform, data centres should have the ability to host each other's data. For example, if some particular data at Parkes are frequently retrieved by another data centre, the system will store these data on that data centre instead. Furthermore, if many applications at Parkes need the same data from another data centre, the system will also move those data to Parkes for storage.

The Parkes Radio Telescope was setup in 1961. For over 40 years, the Parkes cluster has accumulated a large amount of data resources in different formats and sizes. Normally, data can be moved to other data centres, but if the size of the data is very large, moving them via the Internet will be inefficient. To transport terabytes of data, the most efficient way is for a delivery company to ship the hard disks [5]. If an application needs the majority of its data from Parkes, it is preferable that it is executed locally and retrieves data from elsewhere. For example, some research projects may need to process the raw data recorded from the telescope by APSR, in order to get some specific results.

3.2. Problem Analysis

Scientific cloud workflows run on the cloud platform, which is composed of many distributed data centres on the Internet (like Parkes cluster) and each connection between data centres has limited bandwidth. Tasks sometimes need to process more than one dataset that may be stored in different data centres. Because of the bandwidth constraints, the movement of datasets between data centres would be the bottle-neck of the system. In [26], the authors proposed a new protocol for data transportation that could provide gigabits of bandwidth. However, it has not been widely supported by the Internet. The popular cloud systems, such as Amazon EC2 [1], still have limited bandwidth [34]. It charges \$0.10 to \$0.15 per gigabyte to move data in to and out of Amazon Web Services over the Internet. Another approach to deal with the bottle-neck of large data transfer is to divide the tasks, i.e. for the tasks that need to process many distributed datasets, we split them to many smaller and parallel sub-tasks, and schedule them to different datasets. Map-Reduce technology [16] is a typical and successful paradigm. It gains great success in the Google File System and Hadoop, as well as in scientific applications [36]. However, Map-Reduce is more applicable to be used within one data centre, since it needs huge interconnected bandwidth, such as the shuffle step that occurs between the Map procedure and the Reduce procedure. Furthermore, in scientific applications, many tasks must use more than one datasets together and can not be further divided, such as the All-Pairs problem [38]. Therefore, data movement is inevitable. In light of this, we have to place the datasets that are needed by the same task in the same data centre as much as possible, so as to minimise data movement when the task is executed. The placement of datasets among data centres is not trivial.

Normally, a cloud computing system needs to decide in which data centres the application data are stored. Most datasets are flexible about where they are stored since they are independent of users. The cloud computing system can automatically store the application data based on some data placement strategies. However, in scientific cloud workflow systems, some data are not such flexible. They have to be stored in some particular data centres due to different reasons. Some common scenarios are demonstrated below.

First, some data may need to be processed by special equipment. In some scientific projects, many special types of equipment are utilised. Some data can only be processed by particular equipment since they are in certain formats, e.g. the signal from Parkes Radio Telescope can only be processed by the equipment at Parkes, such as the ASPR. These data have to be stored where the required equipment is located.

Second, some data are naturally distributed and too large to be moved efficiently. For example, the raw data files recorded by ASPR are usually terabytes or even petabytes in size. They are naturally stored in Parkes, and impossible to move to other locations via the Internet.

Another reason that some data must be placed at a particular data centre is about the ownership. Data are considered as an important and valuable resource in many scientific projects. The cloud computing platform offers a new paradigm for cooperation that institutions can easily share their valuable data resources by placing a charge on them. So the data with limited access rights have to be stored in particular data centres.

No matter what the reason that the data must be stored in a particular data centre, we call these datasets as *fixed location datasets* in general. As such, we call the datasets that the system can flexibly decide where to store *flexible location datasets*. The data placement strategy not only has to place the flexible location datasets, but also has to take into account the impact of the fixed location datasets. Some challenges exist in the data placement strategy as discussed below.

First, in scientific workflows, both tasks and datasets could be numerous and make up a complicated many-to-many relationship. One task might need many datasets and one dataset might be needed by many tasks. Furthermore, new datasets will be generated during the workflow execution. One dataset generated by a task might be used by several later tasks. So the data placement strategy should be based on these data dependencies.

Second, the scientific cloud workflow system is a dynamic computing environment. Many workflow instances will run in the system simultaneously. Some instances might need long time execution and some might be short. New workflow instances could deploy to the system and completed instances could be removed from the system anytime. So the relationships between datasets and tasks will change often and the placement of datasets has to be changed accordingly.

Third, the data management in scientific cloud workflow systems is opaque to users, that means users do not know where and how the data been stored. In the cloud environment, users only pay for the computation and storage resources that they need and give the application data to the system for processing. Because the cloud systems are built on the service oriented architecture (SOA), the users just use the dynamic cloud services and do not know the infrastructure of the system. Hence, the data placement has to be automatic.

4. BASIC STRATEGIES FOR DATA PLACEMENT

For scientific workflow data management, there are two types of data we have to deal with.

First is the *existing data* that exists before the workflow execution starts. This type of data mainly includes the resource data from the existing file systems or databases and the application data from users as input for processing or analysis.

Second is the *generated data* that are generated during the workflow execution. This type of data mainly includes the newly generated mediate and result data, as well as the streaming data dynamically collected from scientific devices during the workflow execution.

We propose this taxonomy because we will treat these two types of data at the workflow build-time and runtime respectively with different algorithms. This taxonomy only indicates the generation time of the datasets. When the generated data moves to a data centre and is stored, it becomes existing data. The most important common feature is that both types of data might be very large. They can not and should not be stored and moved wherever and whenever we want, since the cloud system has the bandwidth constraints.

The application data of scientific workflow could also have a variety of formats (e.g. XML data, complex objects, raw data files, tables in relational databases). But in this paper, we do not consider the structure of the data, since it is not the main focus of this paper and we will treat all data in the same way.

In scientific workflows, moving data to one data centre will cost more than scheduling tasks to that centre [3]. Hence, our basic strategy is to have a reasonable placement of data in distributed data centres first, so that when tasks are scheduled to the appropriate data centres, almost all the datasets they need are in local storage. In this work we analyse the dependencies between datasets. Based on this dependency, we adapt the *k*-means clustering algorithm to cluster datasets to the proper data centres.

In scientific cloud workflow systems, many workflow instances will run simultaneously, each of which have complex structures. Large numbers of tasks will access large numbers of datasets and produce large output data. In order to execute a task, all required datasets must be located on the same data centre, and this may require some movement of datasets. Furthermore, if two datasets are always used together by many tasks, they should be stored together in order to reduce the frequency of data movement. Here, we say that these two datasets have dependency. In other words, two datasets are said to be dependent on each other if they are both used by the same task. The more tasks there are that use the same datasets, the higher the dependency between those datasets. We denote the set of datasets as D and the set of tasks as T.[2] To represent this dependency, we give every dataset a task set in addition to its size. So, every dataset $d_i \in D$ has two attributes denoted as $<T_i, s_i>$, where $T_i \subset T$ is the set of tasks that will use dataset d_i, s_i denotes the size of d_i. Furthermore, we use *dependency*$_{ij}$ to denote the dependency between datasets d_i and d_j. We say that the datasets d_i and d_j have dependency if there are tasks that will use d_i and d_j together and the quantity of this dependency is the number of tasks that use both d_i and d_j.

[2] All the denotations are listed at the end of the paper.

$$dependency_{ij} = Count(T_i \cap T_j)$$

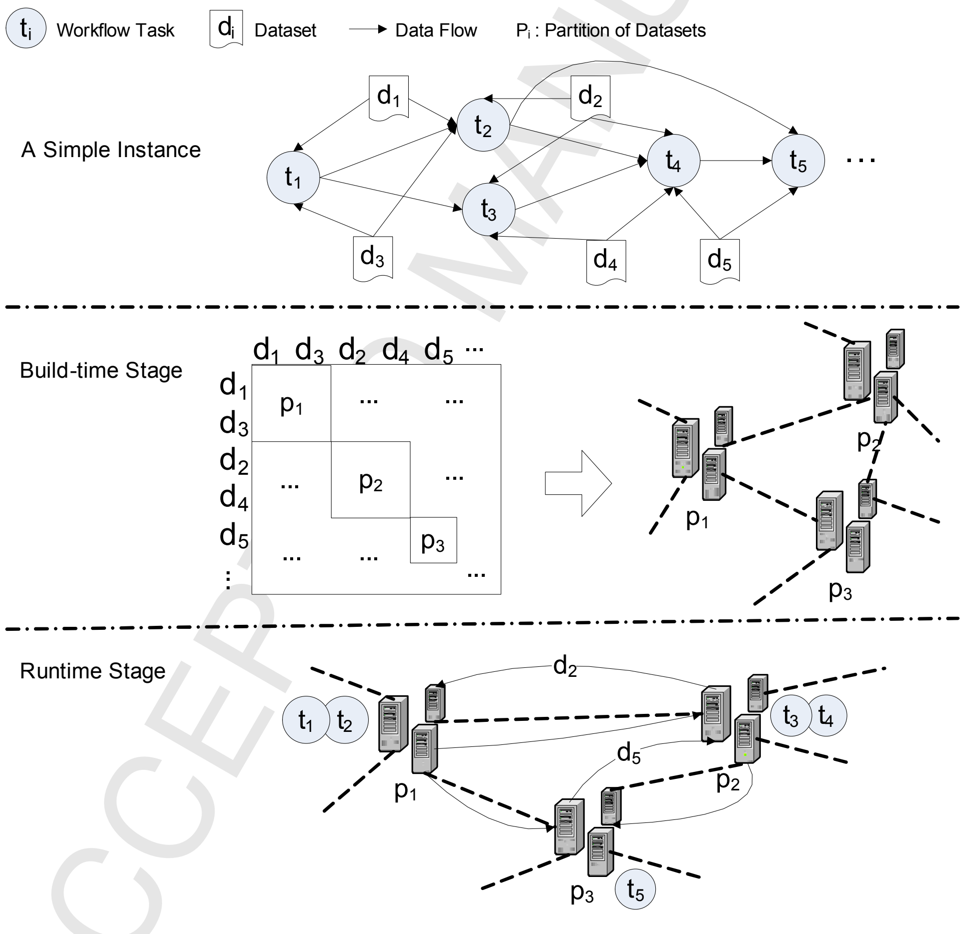
In this work, our *k*-means clustering data placement strategy is based on this dependency that can cluster the datasets into different data centres. The strategy has two stages: build-time and runtime.

At the build-time stage, the main goal of the algorithm is to set up *k* initial partitions for the *k*-means algorithm. We use a matrix based approach to cluster the existing datasets into *k* data centres as the initial partitions.

At the runtime stage, the main goal of the algorithm is to cluster the newly generated datasets to one of the *k* data centres based on their dependencies, which will be calculated dynamically.

We have to design different algorithms for build-time and runtime stages to treat the existing data and generated data respectively, mainly because of the dynamic nature of the cloud environment. Even though we know the size and related tasks of the datasets that will be generated during the workflow execution, it is not practical to calculate their dependencies and assign them a data centre at build-time stage. This is because the scientific workflows have a large number of tasks and need a long time for execution. It is very hard to predict when a certain dataset will be generated in a dynamic cloud environment. If we assign the generated data a data centre at the build-time stage, then when the data are actually generated the data centre might have not enough available storage to store them. Furthermore, it is impractical and inefficient to reserve the storage for the generated data at the build-time stage. This is because the data might not be generated until the end of the scientific workflow and it would be a waste of the reserved storage space during this time.

5. MATRIX BASED *K*-MEANS CLUSTERING STRATEGY FOR DATA PLACEMENT



t_i Workflow Task
d_i Dataset
Data Flow
P_i : Partition of Datasets
A Simple Instance
d_1 d_2 d_3 d_4 d_5
t_1 t_2 t_3 t_4 t_5 …
Build-time Stage
d_1 d_3 d_2 d_4 d_5 …
p_1 p_2 p_3
Runtime Stage
d_2 d_5
t_1 t_2 t_3 t_4 t_5
p_1 p_2 p_3

Figure 1. Example of data placement

In this section we will intricately discuss our data placement strategy. In Fig. 1, there is an example of a simple workflow instance, and it shows the two stages of our strategy. The data flows in the workflow instance, for example, from dataset d_1 to tasks t_1 and t_2 mean that d_1 will be used by both t_1 and t_2; and data flows from t_1 to t_2 and t_3 mean that the dataset generated by t_1 will be used by both t_2 and t_3. During the build-time stage, we partition the existing datasets into several partitions, denoted as $p_1, p_2 ... p_n$, based on their dependencies, and distribute these partitions into different data centres. During the runtime stage, tasks may retrieve datasets from other data centres as needed, and we also pre-allocate generated datasets to the appropriate data centres.

5.1. Build-Time Stage Algorithm

During the build-time stage, we use a matrix model to represent the existing data. We pre-cluster the datasets by transforming the matrix, and then distributing the datasets to different data centres as the initial partitions for the *k*-means clustering algorithm, to be used during the runtime stage. The build-time stage algorithm has two steps and the pseudocode is shown in Fig. 4.

Step 1: Setup and cluster the dependency matrix.

First, we calculate the data dependencies of all the datasets and build up a dependency matrix *DM* (Line 3 in Fig. 4), where *DM*'s element $DM_{ij} = dependency_{ij}$. $dependency_{ij}$ is the dependency value between datasets d_i and d_j, as we defined in the previous section. It can be calculated by counting the tasks in common between the task sets of d_i and d_j, which are denoted as T_i and T_j. Specially, for the elements in the diagonal of *DM*, each value means the number of tasks that will use this dataset. In our algorithm, *DM* is an $n \times n$ symmetrical matrix where n is the total number of existing datasets. If we take the simple workflow instance in Fig. 1 as an example (with only 5 datasets, namely d_1 to d_5, in the system initially), the dependency matrix *DM* is shown in Fig. 2.

$$\begin{array}{l} d_1.T_1=\{t_1,\ t_2\} \\ d_2.T_2=\{t_2,\ t_3,\ t_4\} \\ d_3.T_3=\{t_1,\ t_2\} \\ d_4.T_4=\{t_3,\ t_4\} \\ d_5.T_5=\{t_4,\ t_5\} \end{array} \quad \xrightarrow[= Count(T_i \cap T_j)]{\begin{array}{l} DM_{ij} \\ = dependency_{ij} \end{array}} \quad DM = \begin{array}{c|ccccc} & d_1 & d_2 & d_3 & d_4 & d_5 \\ \hline d_1 & 2 & 1 & 2 & 0 & 0 \\ d_2 & 1 & 3 & 1 & 2 & 1 \\ d_3 & 2 & 1 & 2 & 0 & 0 \\ d_4 & 0 & 2 & 0 & 2 & 1 \\ d_5 & 0 & 1 & 0 & 1 & 2 \end{array}$$

Figure 2. Build up dependency matrix

The dependency matrix (i.e. *DM*) is dynamically maintained at the runtime. When new datasets are generated by tasks or added to the system by users, we calculate their dependencies with all the existing datasets and add them to *DM*.

Next, we use the BEA (Bond Energy Algorithm) to transform the dependency matrix *DM* (Line 4 in Fig. 4). BEA was proposed in 1972 [37] and has been widely utilised in distributed database systems for the vertical partition of large tables [40]. It is a permutation algorithm that can group the similar items together in the matrix by permuting the rows and columns. In our work, it takes the dependency matrix (*DM*) as input, and generates a clustered dependency matrix (*CM*). In *CM*, the items with similar values are grouped together (i.e. large values with other large values, and small values with other small values). We define a global measure (*GM*) of the dependency matrix:

$$GM = \sum_{i=1}^{n} \sum_{j=1}^{n} DM_{ij} (DM_{i,j-1} + DM_{i,j+1})$$

The permutation is done in such a way as to maximise this measure. The detailed algorithm of permutation could be found in [40]. Fig. 3 shows the *CM* of the example *DM* after the BEA transformation.

$$DM = \begin{array}{cccccc} & d_1 & d_2 & d_3 & d_4 & d_5 \\ d_1 & 2 & 1 & 2 & 0 & 0 \\ d_2 & 1 & 3 & 1 & 2 & 1 \\ d_3 & 2 & 1 & 2 & 0 & 0 \\ d_4 & 0 & 2 & 0 & 2 & 1 \\ d_5 & 0 & 1 & 0 & 1 & 2 \end{array} \quad \xrightarrow{\text{BEA}} \quad CM = \begin{array}{cccccc} & d_1 & d_3 & d_2 & d_4 & d_5 \\ d_1 & 2 & 2 & 1 & 0 & 0 \\ d_3 & 2 & 2 & 1 & 0 & 0 \\ d_2 & 1 & 1 & 3 & 2 & 1 \\ d_4 & 0 & 0 & 2 & 2 & 1 \\ d_5 & 0 & 0 & 1 & 1 & 2 \end{array}$$

Figure 3. BEA transformation of dependency matrix

In this step, we do not consider the difference between fixed location datasets and flexible location datasets. If there are some fixed location datasets in the system, they will be arbitrarily scattered in the columns and rows of the dependency matrix, since we built up the matrix by calculating dependencies between all the datasets. After the BEA transformation, all the datasets, including the fixed location datasets, are clustered by their dependencies.

Build-time Stage Algorithm

Input: *D*: set of existing datasets $d_1, d_2, \dots d_n$
DC: set of data centres $dc_1, dc_2, \dots dc_m$
Output: *K*: set of data centres with initial datasets

```
01. K=Ø; FP=Ø; NFP=Ø;       //Initialization. FP: set of partitions that have fixed location datasets
                            //NFP: set of partitions that have not fixed location dataset
02. For (every $dc_i$ in DC) $i\_cs_i = cs_i * \lambda_{ini}$ ;   //Calculate initial available storage of all data centres
03. DM = $dependency_{ij} = Count (T_i \cap T_j)$ ;   //Step 1: setup DM
04. CM = BEA (DM) ;                 //Step 1: BEA transformation
05. if (CM contains fd)             //Step 2 starts. Check the existence of fixed location datasets
06.   Partition&Classify (CM)       //Sub-step 1: partition CM and classify the partitions in to FP and NFP
07.     if ($CM_T$ contains fd & the fd belong to different dc)
08.       Partition&Classify ($CM_T$) ;   //Recursively partition and classify $CM_T$
09.     else if ($CM_T$ contains fd)
10.       add $CM_T$ to FP ;        //$CM_T$ has fixed location datasets, add to FP
11.     else add $CM_T$ to NFP ;    //$CM_T$ has not fixed location datasets, add to NFP
12.     if ($CM_B$ contains fd & the fd belong to different dc)
13.       Partition&Classify ($CM_B$) ;   //Recursively partition and classify $CM_B$
14.     else if ($CM_B$ contains fd)
15.       add $CM_B$ to FP ;        //$CM_B$ has fixed location datasets, add to FP
16.     else add $CM_B$ to NFP ;    //$CM_B$ has not fixed location datasets, add to NFP
17.   for (every data centre $dc_i$ in DC)   //Sub-step 2: distribute the partitions with fixed location datasets
18.     if ($dc_i$ has fd)          //Choose the data centre $dc_i$ that has fixed location datasets
19.       for (every $fd_j$ in $FD_i$)   //Go through all the fixed location datasets belong to $dc_i$
20.         find $CM_j$ in FP ;     //Pick out the partitions that contain these fixed location datasets from PF
21.         add $CM_j$ to $P_i$ ;   //Setup the partitions set P for $dc_i$
22.       calculate $ps_i = \sum_{cm_j \in P_i} ds_j$ ;   //The total size of the partitions in P
23.       while ($ps_i > i\_cs_i$)  //Further partition if the size of P is too large for $dc_i$
24.         find $CM_k$ in $P_i$ , where $ds_k = \max_{cm_i \in P_i} ds_i$ ;   //Largest partition in P
25.         remove $CM_k$ from $P_i$;
26.         BinaryPartition ($CM_k$) ;   //Partition $CM_k$ and update the partitions sets
27.         if ($CM_{kT}$ contains fd) add $CM_{kT}$ to $P_i$;
28.         else add $CM_{kT}$ to NFP ;
29.         if ($CM_{kB}$ contains fd) add $CM_{kB}$ to $P_i$;
30.         else add $CM_{kB}$ to NFP ;
31.         calculate $ps_i = \sum_{cm_j \in P_i} ds_j$ ;   //New size of P after partition
32.       distribute all $CM_j$ in $P_i$ to $dc_i$;   //Distribute datasets
33.       update $dc_i$ to K ;
34.       $i\_cs_i = i\_cs_i - ps_i$ ;
35. else add CM to NFP ;            //CM do not contain fixed location datasets
36. for (all the partitions $CM_i$ in NFP)   //Sub-step 3: distribute the partitions without fixed location datasets
37.   Partition&Distribute ($CM_i$)   //Partition and distribute $CM_i$
38.     if ($ds_T < \max_{j=1}^{m} cs_j$ )   //Size of $CM_{iT}$ is small enough for some data centres
39.       find $dc_j$ from DC,      //Find the best data centre
40.         where $cs_i = \min_{j=1}^{m} \left( cs_j > ds_T \right)$ ;
41.       distribute $CM_{iT}$ to $dc_j$ ;   //Distribute datasets
42.       update $dc_j$ to K ;
43.       $i\_cs_j = i\_cs_j - ds_{iT}$;
44.     else Partition&Distribute ($CM_{iT}$) ;   //Recursively partition and distribute $CM_{iT}$
45.     if ($ds_B < \max_{j=1}^{m} cs_j$ )   //Size of $CM_{iB}$ is small enough for some data centres
46.       find $dc_j$ from DC,      //Find the best data centre
47.         where $cs_i = \min_{j=1}^{m} \left( cs_j > ds_B \right)$;
48.       distribute $CM_{iB}$ to $dc_j$;   //Distribute datasets
49.       update $dc_j$ to K ;
50.       $i\_cs_j = i\_cs_j - ds_{iB}$;
51.     else Partition&Distribute ($CM_{iB}$) ;   //Recursively partition and distribute $CM_{iB}$
52. Return K ;
```

Figure 4. Build-time stage algorithm

Step 2: Partition and distribute datasets.

In this step we will distribute the datasets to data centres as the initial *k* partitions for the *k*-means clustering algorithm at the runtime stage. We denote the set of data centres as *DC*. As shown in Fig. 1, we partition the clustered dependency matrix and place the corresponding datasets to different data centres. However, each dataset d_i has a size s_i and each data centre dc_j also has a storage capacity denoted as cs_j. To find the best partitioning of datasets matching the data centres' storage is an NP-hard problem, since it could be reduced to the Knapsack Packing Problem. Here, we develop a recursive binary partitioning algorithm to find the approximate best solution.

First, we partition *CM* into two parts $\{d_1, d_2 \ldots d_p\}$ and $\{d_{p+1}, d_{p+2} \ldots d_n\}$, which maximises the following measurement:

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

This measurement, *PM,* means that datasets in each partition have higher dependencies with each other and lower dependencies with the datasets in the other partitions. Based on this measure we can simply calculate all *PMs* for $p=1, 2 \ldots n-1$, and choose *p* such that it has the maximum *PM* value as the partition point.

After one partition, the *CM* forms two new clustered matrices, we denote the top one as CM_T, which contains the dependencies of datasets $D_T = \{d_1, d_2 \ldots d_p\}$ and the bottom one as CM_B, which contains the dependencies of datasets $D_B = \{d_{p+1}, d_{p+2} \ldots d_n\}$. Every clustered matrix represents a partition of datasets and we denote the total size of the datasets it contains as $ds = \sum_{i=1}^{n} s_i$. Hence the *ds* for CM_T and CM_B are $ds_T = \sum_{i=1}^{p} s_i$ and $ds_B = \sum_{i=p+1}^{n} s_i$ respectively.

Next, we distribute datasets to data centres by recursively partitioning the clustered dependency matrix.

For each of the data centres, we introduce a percentage parameter λ_{ini} to denote the initial usage of their storage capacity, which means that the initial size of datasets in data centre dc_i could not exceed $cs_i * \lambda_{ini}$. The reason we can not fill the data centre with their maximum storage is that in scientific workflows, the generated data can also be very large. We have to reserve sufficient space in data centres to store those data during the workflow execution. λ_{ini} is an experience parameter. The value of λ_{ini} should depend on what kinds of applications are running on the system, because the generated data of different applications might have different sizes. Furthermore, we also assume that the data centres can host all the application data in the system, i.e. $\sum_{i=1}^{n} s_i < \sum_{i=1}^{m}\left(cs_i * \lambda_{ini}\right)$.

To distribute the datasets, we have to examine whether there are fixed location datasets in the system (Line 5 in Fig. 4). If the system does not have fixed location datasets (Line 35 in Fig. 4), we will recursively partition the sub-matrices CM_T and CM_B until the size of the sub-matrix can fit into one of the data centres' initial storage size limits ($ds <= cs_i * \lambda_{ini}$). Then we distribute the datasets in this sub-matrix into this data centre, and add the reference of this data centre (dc_i) to *K*, where *K* is a set of data centres. When the partitioning of *CM* finishes, all the initial datasets are moved to proper data centres. We take the data centres in *K* as the initial partitions of the *k*-means clustering algorithm.

If there are fixed location datasets in the system, the distribution process is more complicated. For a fixed location dataset fd_i, we denote it as $<T_i, s_i, dc>$, where the additional attribute *dc* is the data centre where this dataset has to be stored. And we use *FD* to denote the set of the fixed location datasets a data centre has. For a data centre that does not have fixed location datasets, *FD* is empty. The distribution is conducted as the three following sub-steps.

Sub-step 1 (Line 6-16 in Fig. 4), we classify fixed location datasets and flexible location datasets in different partitions. We also need to recursively partition the sub-matrices CM_T and CM_B. The stop condition is that the sub-matrix does not have fixed location datasets or all the fixed location datasets it has belong to one data centre. We add the partitions that do not have fixed location datasets to a set named *NFP* and the partitions have fixed location datasets to a set named *FP*.

Sub-step 2 (Line 17-34 in Fig. 4), we distribute the partitions with fixed location datasets in *FP*. We need to check the data centres' information. For the data centres that have fixed location datasets, we pick out the partitions that contain these fixed location datasets from *FP*, denote as *P*. Then, we calculate the total size of these partitions, denote as *ps*, where $ps = \sum_{CM_i \in P} ds_i$. If these partitions can fit into this data centre, we store them. If not, we recursively pick the largest partition from P, binary partition it and move the part that does not have fixed location datasets to NFP, until these partitions can fit into the data centre.

Sub-step 3 (Line 36-51 in Fig. 4), we distribute the partitions that only contain flexible location datasets in *NFP*. We start with the largest one and go through all the partitions in *NFP* by their size. For every partition, we distribute it to the data centres by recursive binary partitioning.

5.2. Runtime Stage Algorithm

At the runtime stage, we use the *k*-means clustering algorithm to dynamically cluster the generated data to one of the *k* data centres based on their dependencies. And when new workflows are deployed to the system or some data centres become overloaded, we also have to adjust the data placement among data centres. The pseudocode of the runtime stage algorithm is shown in Fig. 5.

For the generated data, some of them could be valuable resources that can be utilised by other workflows, but most of them are temporal data. They are generated by the preceding tasks in the workflows and will be used by the subsequent tasks. They do not need permanent storage and will be deleted after the workflows have finish execution. In many scientific applications, the temporal data are in large volumes [19]. Some researches demonstrated that timely removal of these temporal data can save a lot of runtime storage space [42]. In our work, we dynamically check and delete the obsolete temporal data before every round of task scheduling. The runtime stage algorithm contains the following two steps.

Step 1: Data pre-allocation by the clustering algorithm.

In this step, the first thing we have to do is task scheduling (Line 2-3 in Fig. 5). Scheduling is a very important issue in scientific workflow systems, especially for computation intensive and/or data intensive applications. Much research has been done into scheduling workflows [43] [52]. However, task scheduling is not the main focus of this paper. Therefore, our scheduling strategy is quite straight forward. We just follow the philosophy of “moving data to a data centre will cost more than scheduling tasks to that centre”, and schedule tasks based on the placement of datasets. We periodically monitor the state of all the workflow tasks and dynamically schedule the ready tasks to the data centre which has the most datasets they require. Here, a task is ready if all the datasets it needs are existing data (i.e. have been generated).

When tasks have been executed, new datasets will be generated. The system will then decide where to put these datasets: either store them locally or allocate them to other data centres. In our work, the system will cluster the newly generated datasets to the data centre that has the highest dependency with them (Line 4-12 in Fig. 5). We define the dependency between dataset d_i and data centre dc_j as dc_dep_{ij}, which is the sum of the dependencies of d_i with all the datasets in dc_j.

Suppose d_u is a new generated dataset and T_u is the set of tasks that will use d_u. First, we calculate the dependencies of d_u with all other datasets in the system and add the new row and column to *DM* for d_u, where

$$DM_{ui} = DM_{iu} = dependency_{ui} = Count\{T_u \cap T_i\} \quad i = 1,2,...n$$

Then we calculate the dependencies of d_u with all the *k* data centres, where

$$dc\_dep_{uj} = \sum_{d_m \in dc_j} dependency_{um}, \quad j = 1,2,...k$$

With these dependencies, we will select the data centre dc_h that has the highest dependency with d_u, where

$$dc\_dep_{uh} = \max_{j=1}^{k}(dc\_dep_{uj})$$

dc_h is the data centre in which we will store the dataset d_u. And we will check the available storage of dc_h, before we move d_u to it.

Here we will introduce a maximum storage usage parameter λ_{max} for data centres, which is a percentage threshold indicating whether a data centre is overloaded or not. λ_{max} is also an experience parameter, just like the initial storage usage parameter λ_{ini}. Hence, the storage that the runtime data can use of a data centre dc_i is $cs_i*(\lambda_{max}-\lambda_{ini})$. The value of λ_{max} depends on the overall workload of the system. If the system workload is heavy, λ_{max} has to be set to a larger value. Likewise, if the system workload is light, λ_{max} is set smaller to prevent too many datasets gathering in one data centre.

We will move the new generated dataset d_u to the selected data centre dc_h, if $cs_h\lambda + s_u < cs_h\lambda_{\max}$ is true, where s_u is the size of d_u and λ is the current storage usage percentage of dc_h. Otherwise, we go to the next step to adjust the data placement.

```
Run-time Stage Algorithm

Input:   T: set of tasks;
         DC: set of data centres;
Output:  All the tasks are finished;

01. for (every ready task $t_i \in T$)       //Step 1: data pre-stage by clustering algorithm
02.   schedule $t_i$ to $dc_j$ to execute,    //Tasks scheduling
03.     where $dc_j$ has the most data sets $t_i$ needs;
04.   for (every generated data set $d_u$ of $t_i$)    //New datasets generated
05.     add $d_u$ to DM,                  //Update dependency matrix
06.       where $dependency_{ui} = Count\{T_u \cap T_i\}, i = 1,2,...n$ ;
07.     for (every data centre $dc_k \in DC$)          //Calculate dependency
08.       calculate $dc\_dep_{uk} = \sum_{d_m \in dc_k} dependency_{um}$ ;
09.     choose $dc_h$ ,                    //Choose data centre and pre-stage data
10.       where $dc\_dep_{uh} = \max_{j=1}^{k}(dc\_dep_{uj})$ ;
11.     if ($cs_h\lambda + s_u < cs_h\lambda_{\max}$ )
12.       move $d_u$ to $dc_h$;
13.     else                //Step 2: adjust data placement
14.       do adjustment
15.         CM'= BEA (DM);
16.         K'= Build-time Algorithm Step 2 ;   //Get new data placement
17.         for (all the dc in DC)
18.           choose $dc_i$,             //Choose the most overloaded data centre
19.             where $\lambda_i = \max_{dc_j \in DC} \lambda_j$;
20.           Compare&Adjust ($dc_i$, $dc_i'$)      //$dc_i'$ is the reference in K'
21.             for (all $d_j \in dc_i \wedge d_j \notin dc_i'$)
22.               send $d_j$;           //Send the datasets belong to other data centres
23.             for (all $d_j \notin dc_i \wedge d_j \in dc_i'$)
24.               retrieve $d_j$;        //Retrieve the datasets it should have
25.   update T for new ready tasks with $d_u$;      //For next round scheduling
```

Figure 5. Runtime stage algorithm

Step 2: Adjust data placement among data centres.

During workflow execution, there are two situations that trigger the need to adjust the data placement among data centres.

The first is when the selected destination data centre dc_h for the new generated dataset does not have enough available storage. This means that dc_h is overloaded. Hence, we have to adjust the datasets placement to balance the overall workload of the system.

The second is when new workflows are deployed to the system. Together with the new workflows, new datasets and tasks will be added to the system. The dependencies of the original datasets will change, since the new tasks might use the existing data in the system. In this situation, we will calculate the dependencies between the new datasets and the existing datasets, and add them to the dependency matrix *DM*. If there are any new tasks which use existing data, they will be added to the task set of the appropriate existing dataset. For every new dataset, we will find an appropriate data centre for it by following the procedure in step 1. If the selected data centre is overloaded, we have to adjust the datasets placement to balance the overall workload of the system.

To adjust the data placement, we need to run some functions from the build-time stage algorithms (Line 15-16 in Fig. 5). First, we do the BEA transformation to cluster the updated dependency matrix (*DM*) and get a new clustered dependency matrix (*CM'*). Next, we run the algorithm in step 2 of the build-time stage, but without actual data distribution. We just calculate the new placement of datasets in the data centres and save the references in a new set of data centres, denoted as *K'*.

Then we can do the adjustment by comparing the old data placement with the new one in *K'* (Line 17-24 in Fig. 5). We start the adjustment from the data centre that has the highest storage load and go through all the data centres by the storage usage in the decreasing order. For every data centre, we compare the datasets it currently has with the new datasets in *K'*. Then we send the datasets that do not belong to this data centre to the ones they now belong to and retrieve the datasets it should have from other data centres.

Since λ_{max} represents a percentage of a data centre's total storage space, each data centre will still have some storage available (100% - λ_{max}) to facilitate data movement during this redistribution. In the case that λ_{max} is set to 100%, additional temporary storage space may need to be acquired to serve as a buffer before the adjustment process can be completed. However, this situation rarely happens in the system, due to the following reasons: 1) in the adjustment process we always select the data centre with the highest storage usage to adjust as the priority, and send its datasets to other data centres first; 2) the total size of the datasets in the system is smaller than the total size of the available storage of all the data centres ($\sum_{i=1}^{n} s_i < \sum_{i=1}^{m} (cs_i * \lambda_{ini})$), because we have the assumption that the data centres can host all the application data in the system; and 3) for every data centre we reserve some storage for the runtime generated datasets ($cs * (\lambda_{\max} - \lambda_{ini})$), this storage space is not always highly utilised, because we delete obsolete datasets dynamically. In our system, for every data centre, we reserve runtime storage for generated datasets as 40% of the initial storage for existing datasets i.e. $(\lambda_{\max} - \lambda_{ini}) / \lambda_{ini} = 40\%$. As addressed in section 6 later, we have run tens of thousands of workflow instances for simulation, and a situation where we lacked storage for data reallocation did not occur.

The data placement strategy in this section states that when a task is scheduled to one data centre during workflow execution, that data centre will have most input datasets for that task. Then, only a small number of datasets have to be retrieved from remote data centres. The simulations in the next section will show that our data placement strategy can greatly reduce the total data movement during workflow execution.

6. SIMULATION

6.1. *Simulation Environment: SwinDeW-C*

SwinDeW-C (Swinburne Decentralised Workflow for Cloud) [52] is developed based on SwinDeW [50] and SwinDeW-G [51]. It is currently running at Swinburne University of Technology, which is composed of 10 servers and 10 high-end PCs. To simulate the cloud computing environment, we set up VMware [4] software on the physical servers and create virtual clusters as data centres. Fig. 6 shows our simulation environment.

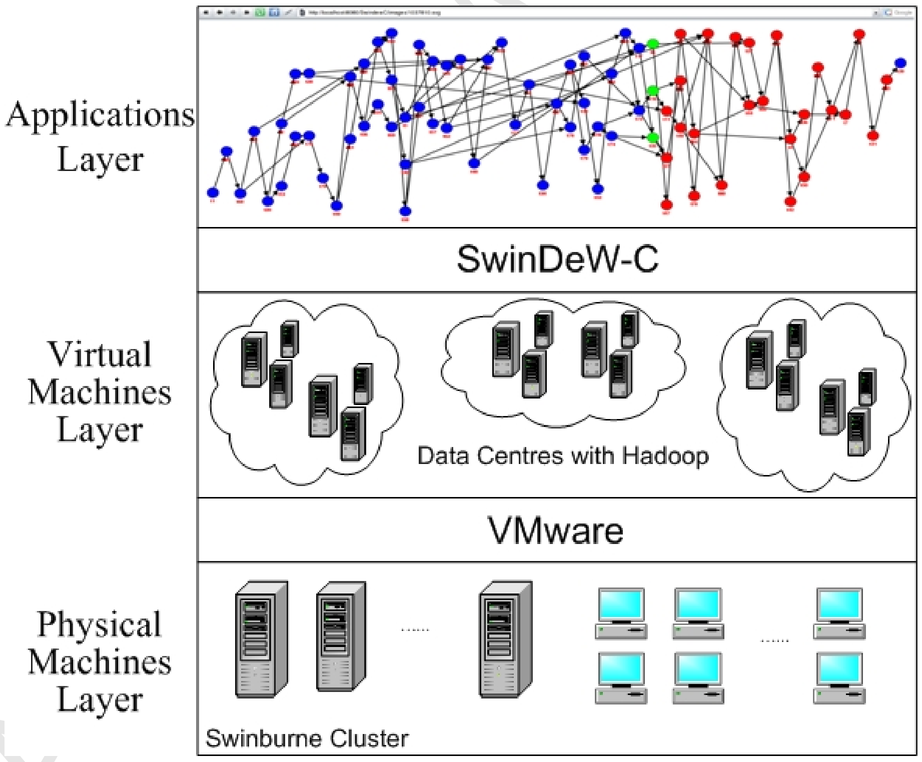


Figure 6. Simulation environment of SwinDeW-C

Every data centre created is composed of 8 virtual computing nodes with storages, and we deploy an independent Hadoop file system on each data centre. SwinDeW-C runs on these virtual data centres that can send and retrieve data to and from each other. Through a user interface at the applications layer, which is a Web based portal, we can deploy workflows and upload application data.

SwinDeW-C is designed for large scale cloud applications. It has a novel architecture for the cloud computing environment. However, the presentation of the comprehensive system design of SwinDeW-C is not the main focus of this paper. In Fig. 7, we only illustrate the key system components of SwinDeW-C that relate to the data placement strategy.

User Interface Module:
The cloud computing platform is built on the Internet and a Web browser is normally the only software needed at the client side. This interface is a Web portal by which users can visit the system and deploy their applications. The *Uploading Component* is for users to upload application data and workflows, and the *Monitoring Component* is for users, as well as system administrators to monitor workflow execution.

Data Management Module:
The *Data Placement Component* is the core component of data management in SwinDeW-C that facilitates the algorithms in our data placement strategy. The *Data Catalogue* is used to store the information of applications which, in a service oriented cloud platform, is a registry for the data services. By using the catalogue, the system can locate the data needed. Other components in this module, such as *Data Replication Component, Data Synchronisation Component, Meta-data Repository* and *Provenance Data Collection* are also essential for cloud data management. Since they are not directly related to the data placement strategy, we do not give their details here.

Other Modules:
The *Flow Management Module* has a *Process Repository* that stores all the workflow instances running in the system. The *Task Management Module* has a *Scheduler* that schedules ready tasks to data centres during the runtime stage of the workflows. Furthermore, the *Resource Management Module* keeps the information of the data centres' usage, and can trigger the adjustment process in the data placement strategy. For other components in these modules, as well as other modules in SwinDeW-C, we do not give the details as the work presented here only focuses on the workflow data management.

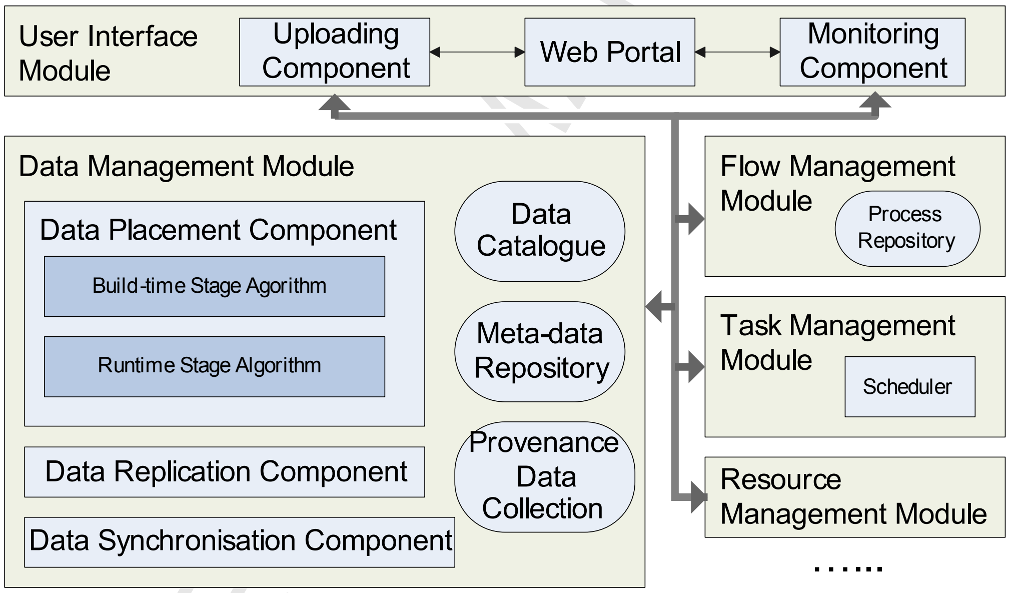


Figure 7. Related key system components of SwinDeW-C

6.2. *Simulation Strategies*

The algorithms in our data placement strategy are for the build-time and runtime stages respectively. To evaluate their performance, we run each workflow instance through 4 simulation strategies:

Random: In this simulation, we randomly place the *existing data* during the build-time stage and store the *generated data* in the local data centre (i.e. where they were generated) at runtime. This simulation represents the traditional data placement strategies in old distributed computing systems (i.e. clusters and early grid systems). At that time, data were usually stored in the local node naturally or in the nodes that had available storages. The temporal intermediate data, i.e. generated data, were also naturally stored where they were generated waiting for the tasks to retrieve them.

Build-time only: This simulation shows the performance of our build-time algorithm. It is used to place the existing data at build-time. During the runtime stage we will store the generated data in the local data centre, as with the Random simulation. In a cloud computing system, data are more flexible than they were in the past; this allows the system can decide where to store them. Our build-time algorithm places the application data based on their dependencies. This simulation will show the data movement reduction in the workflows' execution by using this algorithm.

Runtime only: This simulation shows the performance of the runtime algorithm by randomly placing the existing data at build-time and by pre-allocating the generated data with our runtime algorithm. This simulation represents the strategy that some popular grid scientific workflows used [13]. Their work shows that pre-allocating data to the computing node where the tasks will execute can reduce the total execution time of the workflow. However, this simulation will show that only pre-allocating data at runtime stage can not reduce the data movement in workflow execution.

Build & Run: This simulation shows the overall performance of our algorithms both at build-time and runtime. Our algorithms are specifically designed for scientific cloud workflows. The strategy is based on data dependency and can automatically place existing data; and cluster generated data to the appropriate data centres. Comparisons with other strategies will be made with different aspects to show the performance of our algorithms.

The traditional way to evaluate the performance of a workflow system is to record and compare the execution time [13] [42]. However, in our work we will count the total data movement instead. The execution time could be influenced by other factors beside data management, such as bandwidth, scheduling strategy and I/O speed. Our data placement strategy aims to reduce the data movement between data centres on the Internet. So we directly take the number of datasets that are actually moved during the workflow execution as the measurement to evaluate the performance of the algorithms. In a cloud computing environment with limited bandwidth based on the Internet, if the total data movement has been reduced, the execution time will be reduced correspondingly. Furthermore, the cost of data transfer will also decrease.

To make the evaluation as objective as possible, we generate test workflows randomly to run on SwinDeW-C. This would make the evaluation results independent of any specific applications. As we need to run the build-time and runtime algorithms separately, we set the number of existing datasets and generated datasets to be the same for every test workflow. That means that we have the same number of existing datasets and tasks for every test workflow, and we assume that each task will only generate one dataset. We can control the complexity of the test workflow by changing the number of datasets. Every dataset will be used by a random number of tasks, and tasks that use generated datasets must be executed after the task that generates their input. We can control the complexity of the relationships between the datasets and tasks by changing the range of this random number. Another factor that would have impact on the algorithms is the number of fixed location datasets. We can randomly choose some percentage of datasets from the existing data and randomly select some data centres for them. We will run new simulations to show the impact on performance. Here we have only included graphs of the simulation results. The detailed configuration and result reports of the simulations, as well as the source code can all be found at http://www.swinflow.org/docs/DataPlacement.zip.

6.3. Simulation Results

Fig. 8 shows the data movement when we run workflows with different complexity on different numbers of data centres. We can see the increases in data movement as the workflows become more complex and the number of data centres increases. All the values in the figure are the average of running 1000 test workflows with the same parameters.

In Fig. 8 (a), we ran the test workflows with different complexity on 15 data centres. We used 4 types of test workflows with different numbers of datasets. In Fig. 8 (b), we fixed the test workflows' datasets count to 50, and ran them on different numbers of data centres. Then we changed 10% of the input datasets to fixed location datasets and ran the same simulation again. The results are shown in Fig. 9.

From the results, we could draw the conclusions that 1) the build-time algorithm can effectively reduce the total data movement of the workflow execution; 2) the runtime algorithm does not reduce the total data movement, and even causes more data movement if the existing datasets are placed randomly and 3) with fixed location datasets added to the system, our algorithms can still work very well with performance only degrading slightly. The runtime algorithm does not decrease the data movement because it pre-allocates datasets before scheduling tasks based on their data dependencies. If the existing datasets are randomly placed, the differing dependencies of the data centres are not obvious. The increase in data movement is caused by pre-allocation of datasets to the wrong data centres. However, if the existing datasets were clustered by the build-time algorithm, the performance of the runtime algorithm would be better.

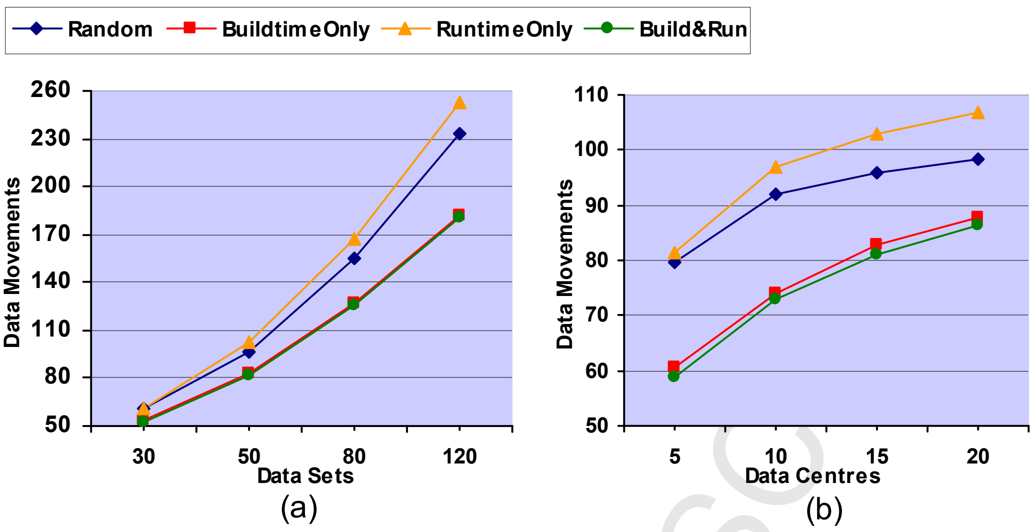


Figure 8. Data movements without runtime storage limit and without fixed location datasets

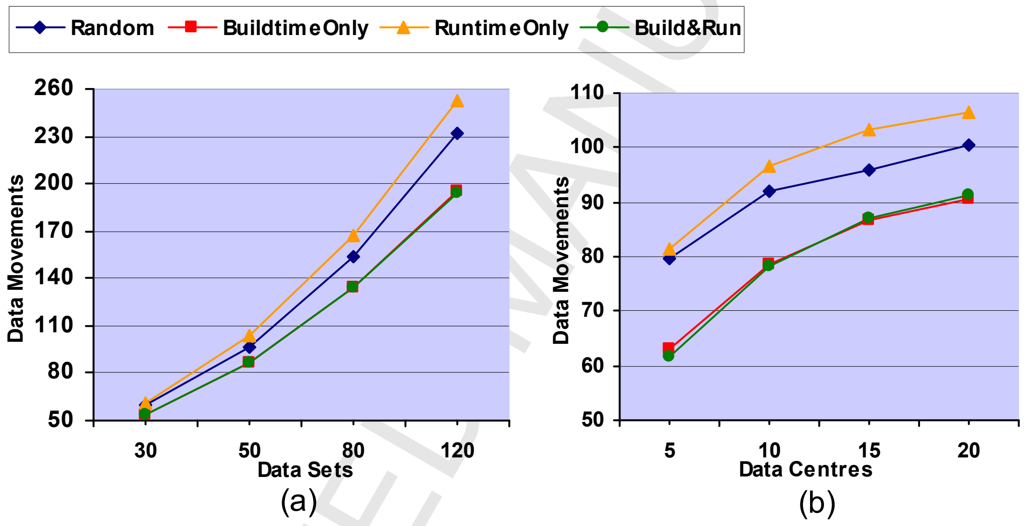


Figure 9. Data movements without runtime storage limit and with 10% of fixed location datasets

However, in the simulation described above, we did not limit the amount of storage that the data centres had available during runtime. The reason for this is that we wanted to see how the tasks and datasets were distributed, which indicates the workload balance among data centres. During the execution of every test workflow instance, we recorded the number of datasets that moved to each data centre, as well as the tasks that scheduled to that data centre. We also calculated the standard deviation of the data centres' usage. Fig. 10 shows the average standard deviation of running 1000 test workflows on 15 data centres each having 80 existing datasets and 80 tasks, both with and without fixed location datasets.

From Fig. 10 we can see relatively high deviations in the data centres' usage in the two simulations without the runtime algorithm. This means that tasks and datasets are allocated to one data centre more frequently. This leads to a data centre becoming a super node that has a high workload. By contrast, in the other two simulations that use the runtime algorithm to pre-allocate the generated data to other data centres, the deviation of data centre usage is low. This demonstrates that the runtime algorithm can make a more balanced distribution of the workload among data centres.

In a cloud computing environment, data centres normally have limited storage, especially in some storage constrained systems. When one data centre is overloaded, we need to reallocate the data to other data centres. The reallocation will not only cause extra data movement, but will also delay the execution of the workflow. To count the reallocated datasets, we ran the same test workflows as in Fig. 10 with a storage limit in every data centre. We limited the runtime storage for generated datasets to 40% of the initial storage for existing datasets i.e. $(\lambda_{\max} - \lambda_{ini})/\lambda_{ini} = 40\%$. In Fig. 11 we show the average data movement including the data reallocation.

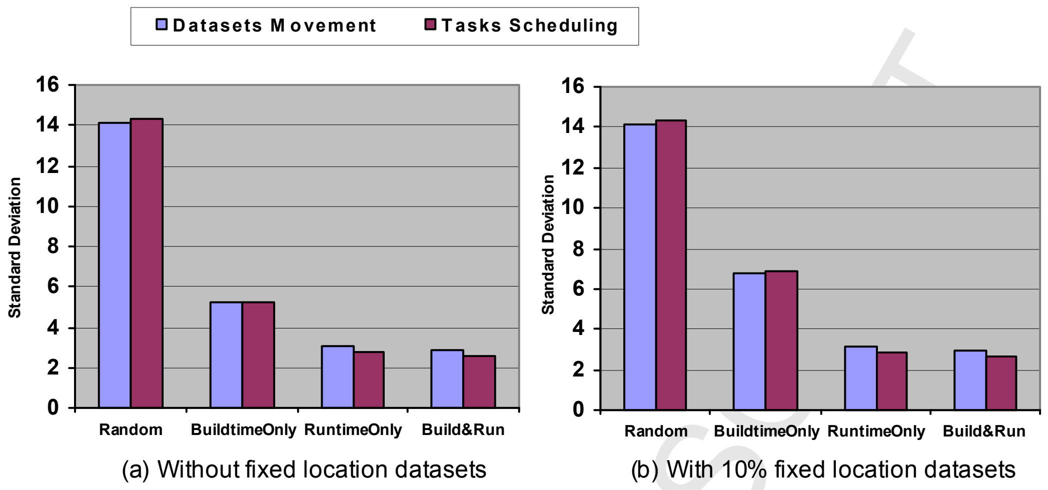


Figure 10. Standard deviation of workload among data centres

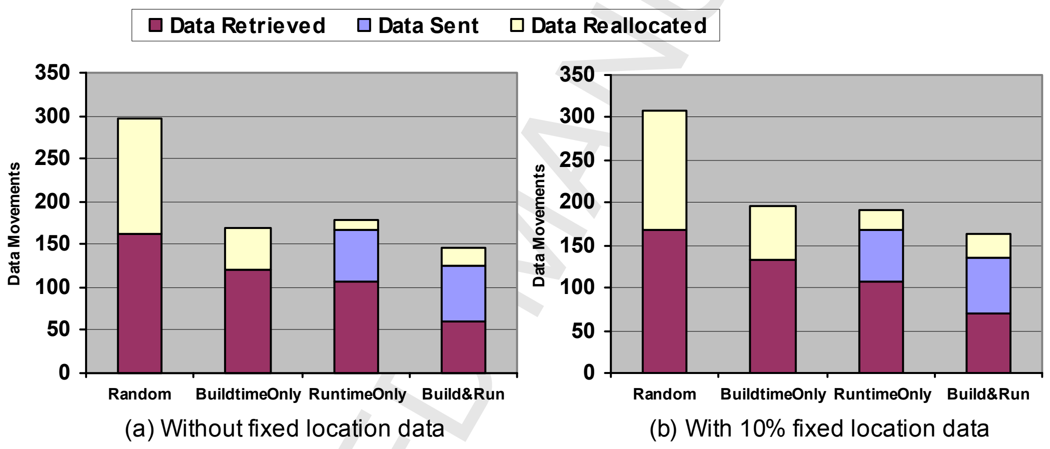
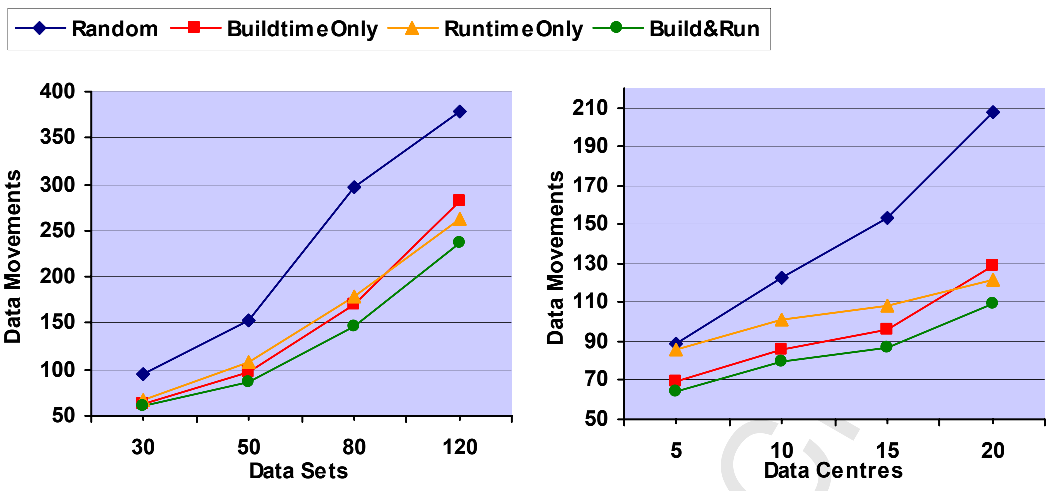


Figure 11. Proportions of 3 types of data movements

From Fig. 11, we can see that a lot of data is reallocated in the simulations without the runtime algorithm. The least data reallocation occurred when we only use the runtime algorithm. However, the least data movement in total occurred when using the build-time and runtime algorithms together. In Fig. 11 (a), using both algorithms caused 146.505 movements of datasets on average. Comparing this to the random simulation, 297.807 datasets movements on average, our algorithms reduce the data movement by 50.8%. On the other hand, the build-time algorithm and runtime algorithm cause movement of 170.26 and 178.662 datasets on average. Compared to the random situation, they reduce the data movements by 42.8% and 40.0% respectively. In Fig. 11 (b), with 10% fixed location datasets in the system, our algorithms (Build&Run) can reduce the data movement by 47.4% compared to the Random simulation.

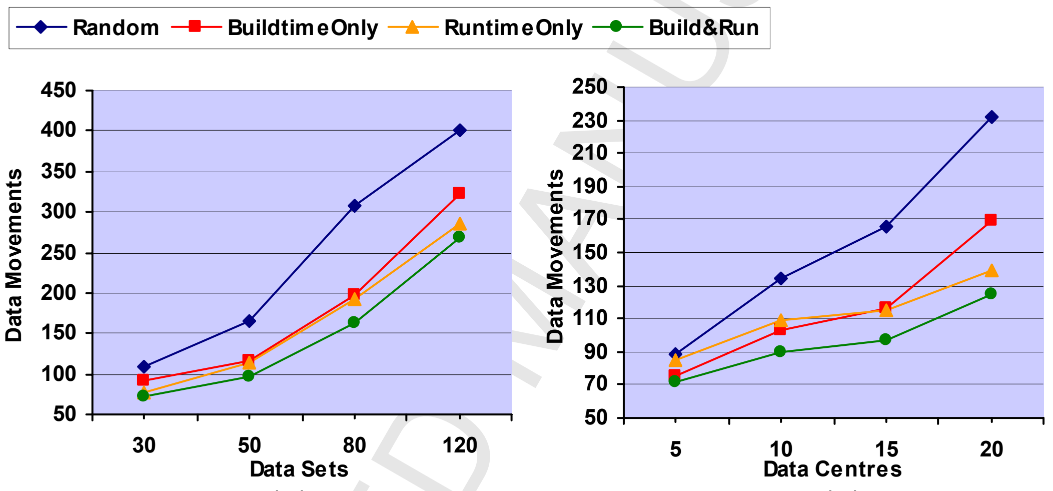
To better evaluate the performance of our algorithms, we give every data centre a runtime storage limit and run the same simulation workflows as Fig. 8. We get the final results of data movement which are shown in Fig. 12.

From Fig. 12 we can see that as the number of data centres and datasets increases, the performance of the build-time algorithm decreases. This is because without the runtime algorithm the datasets and tasks are gathering on the one data centre. This triggers the adjustment process more frequently, which costs extra data movements. Furthermore, we ran the same simulation as Fig. 12 under the condition that the system has fixed location datasets. Fig. 13 shows the data movements when we set the percentage of fixed location datasets to 10%. We can see our algorithms can still reduce the data movements significantly. Furthermore, with higher percentages of fixed location datasets in the system, our algorithms still work, and we will demonstrate this in the next simulation.



(a) (b)

Figure 12. Data movements with runtime storage limit



(a) (b)

Figure 13. Data movements with runtime storage limit and with 10% fixed location datasets

Fig. 13 has consistent results with Fig. 9, that the fixed location datasets have a negative impact on the algorithms' performance. In the algorithms, we try to place the datasets on data centres based on dependencies, however, the fixed location datasets have to be stored in particular data centres. This will decrease performance, as fixed location datasets will prevent the algorithms from placing datasets with their dependencies. However, given the existence of fixed location datasets, our algorithms can still reduce data movement by placing the flexible location datasets with dependencies. To demonstrate the impact of fixed location datasets on the algorithms, we conducted another batch of simulations. We ran 1000 test workflows on 15 data centres each having 80 existing datasets and 80 tasks, but with different percentages of fixed location datasets. As the number of fixed location datasets increases, we can see their impact on data movement in Fig. 14.

From Fig. 14 (a) we can see that as the percentage of fixed location datasets goes up, the data movements of the Build-time only and Build & Run simulations go up accordingly; however the Random and Runtime only simulations keep steady. This means the fixed location datasets primarily have an impact on the build-time algorithm. This is because all the fixed location datasets are existing data, which are placed by the build-time stage algorithm. When the percentage reaches 60%, the data movements of Build & Run simulation even exceeds the Random simulation. This is because the pre-allocation of datasets in the runtime algorithm causes more data movements, as the build-time algorithm gets worse. In Fig. 14 (b) it may seem slightly confusing that the data movements of all simulations go up and then drop, as the percentage of fixed location datasets goes up. This is because when we set the runtime storage limit, many data movements are caused by data reallocation. However, the fixed location datasets are not involved in the overload adjustment process. Hence, the data movement decreases. In this figure we can also see that the fixed location datasets may have a negative impact on the build-time algorithm.

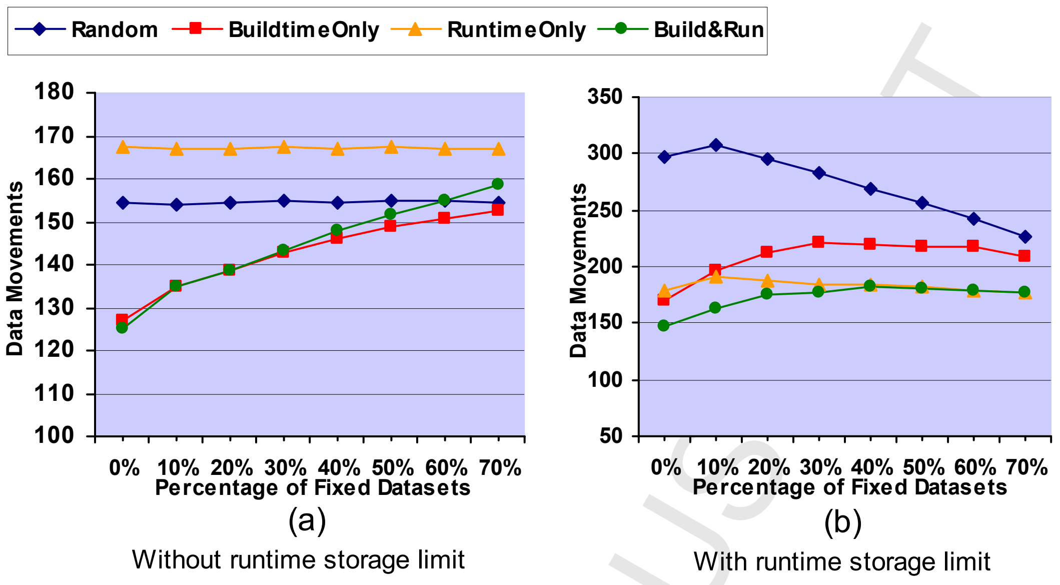


Figure 14. Data movements with different percentage of fixed location datasets

7. Conclusions and Future Work

In this paper, we examined the unique features of scientific cloud workflows and proposed a clustering data placement strategy that can automatically allocate application data among data centres based on dependencies. Simulations in our cloud workflow system SwinDeW-C indicated that our data placement strategy can effectively reduce data movement during workflow execution. The build-time algorithm reduces the amount of data retrieved and the run time algorithm guarantees a balanced distribution of data and can reduce data movement incurred by data reallocation, even when fixed location data exist in the system.

In our current work, to guarantee the data reliability, we used Hadoop's replication mechanism within a data centre, and among data centres we did not use any replication strategies. The data used in scientific workflow applications are usually very large and as such it is not efficient to replicate all the application data in the system. However, replication of frequently used data could also reduce data movement. In the future work, we will develop some efficient replication strategies for the data placement algorithm, which could balance the data movement and storage usage. Furthermore, in our current simulation we measure the reduction of datasets' movements to evaluate our strategy. In the future, we will meter the execution time of the workflow as well, which can better demonstrate the effectiveness of our strategy. To be more comprehensive, we will also incorporate the size of datasets to calculate the data dependency, and adapt some popular cloud service providers' pricing models to our simulation, which will show the cost effectiveness of our strategy.

Acknowledgment

The research work reported in this paper is partly supported by Australian Research Council under Linkage Project LP0990393. We are grateful to Bryce Gibson and Michael Jensen for the accomplishment of the simulation work, as well as the carefull English proofreading.

References

[1] "Amazon Elastic Computing Cloud, http://aws.amazon.com/ec2/", accessed on 25 November 2009.
[2] "ATNF Parkes Swinburne Recorder, http://astronomy.swin.edu.au/pulsar/?topic=apsr", accessed on 25 November 2009.
[3] "Hadoop, http://hadoop.apache.org/", accessed on 25 November 2009.
[4] "VMware, http://www.vmware.com/", accessed on 25 November 2009.
[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," University of California at Berkeley, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, Technical Report UCB/EECS-2009-28, accessed on 25 November 2009.

[6] R. Barga and D. Gannon, "Scientific versus Business Workflows," in *Workflows for e-Science*, pp. 9-16, 2007.

[7] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," in *IBM Centre for Advanced Studies Conference*, Toronto, Canada pp. 1-12, 1998.

[8] M. Brantner, D. Florescuy, D. Graf, D. Kossmann, and T. Kraska, "Building a Database on S3," in *SIGMOD*, Vancouver, BC, Canada, pp. 251-263, 2008.

[9] R. Buyya and S. Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report," in *IEEE International Workshop on Grid Economics and Business Models*, Seoul, pp. 19-66, 2004.

[10] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in *10th IEEE International Conference on High Performance Computing and Communications (HPCC-08)*, Los Alamitos, CA, USA, 2008.

[11] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems,* vol. in press, pp. 1-18, 2009.

[12] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services," in *ACM/IEEE conference on Supercomputing*, Baltimore, Maryland, pp. 1-17, 2002.

[13] A. Chervenak, E. Deelman, M. Livny, M.-H. Su, R. Schuler, S. Bharathi, G. Mehta, and K. Vahi, "Data Placement for Scientific Applications in Distributed Environments," in *8th Grid Computing Conference*, pp. 267-274, 2007.

[14] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang, "Programming scientific and distributed workflow with Triana services," *Concurrency and Computation: Practice and Experience,* vol. 18, pp. 1021-1037, 2006.

[15] J. M. Cope, N. Trebon, H. M. Tufo, and P. Beckman, "Robust data placement in urgent computing environments," in *IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009*, pp. 1-13, 2009.

[16] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM,* vol. 51, pp. 107-113, 2008.

[17] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping Scientific Workflows onto the Grid," in *European Across Grids Conference*, pp. 11-20, 2004.

[18] E. Deelman and A. Chervenak, "Data Management Challenges of Data-Intensive Scientific Workflows," in *IEEE International Symposium on Cluster Computing and the Grid*, pp. 687-692, 2008.

[19] 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,* vol. In Press, Corrected Proof.

[20] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The Cost of Doing Science on the Cloud: the Montage example," in *ACM/IEEE Conference on Supercomputing*, Austin, Texas, pp. 1-12, 2008.

[21] S. Doraimani and A. Iamnitchi, "File grouping for scientific data management: lessons from experimenting with real traces," in *Proceedings of the 17th international symposium on High performance distributed computing* Boston, MA, USA: ACM, 2008, pp. 153-164.

[22] G. Fedak, H. He, and F. Cappello, "BitDew: a programmable environment for large-scale data management and distribution," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Austin, Texas, pp. 1-12, 2008.

[23] I. Foster, Z. Yong, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop, GCE '08*, pp. 1-10, 2008.

[24] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SIGOPS Oper. Syst. Rev.,* vol. 37, pp. 29-43, 2003.

[25] R. Grossman and Y. Gu, "Data Mining Using High Performance Data Clouds: Experimental Studies Using Sector and Sphere," in *SIGKDD*, pp. 920-927, 2008.

[26] R. Grossman, Y. Gu, M. Sabala, and W. Zhang, "Compute and storage clouds using wide area high performance networks," *Future Generation Computer Systems,* pp. 179–183, 2008.

[27] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering,* vol. 15, pp. 515-528, 2003.

[28] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: near-optimal block placement and replication in distributed caches," in *Proceedings of the 36th annual International Symposium on Computer Architecture, ISCA '09*, Austin, TX, USA, pp. 184-195, 2009.

[29] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the Use of Cloud Computing for Scientific Workflows," in *4th IEEE International Conference on e-Science*, pp. 640-645, 2008.

[30] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.,* vol. 31, pp. 264-323, 1999.

[31] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science Clouds: Early Experiences in Cloud Computing for Scientific Applications," in *First Workshop on Cloud Computing and its Applications (CCA'08)*, pp. 1-6, 2008.

[32] T. Kosar and M. Livny, "A framework for reliable and efficient data placement in distributed computing systems," *Journal of Parallel and Distributed Computing,* vol. 65, pp. 1146-1157, 2005.

[33] T. Kosar and M. Livny, "Stork: making data placement a first class citizen in the grid," in *Proceedings of 24th International Conference on Distributed Computing Systems, ICDCS 2004*, pp. 342-349, 2004.

[34] H. Liu and D. Orban, "GridBatch: Cloud Computing for Large-Scale Data-Intensive Batch Applications," in *Eighth IEEE International Symposium on Cluster Computing and the Grid*, pp. 295-305, 2008.

[35] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, and E. A. Lee, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience,* pp. 1039–1065, 2005.

[36] A. Matsunaga, M. Tsugawa, and J. Fortes, "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications," in *4th IEEE International Conference on e-Science*, pp. 222-229, 2008.

[37] W. T. McCormick, P. J. Sehweitzer, and T. W. White, "Problem Decomposition and Data Reorganization by a Clustering Technique," *Operations Research,* vol. 20, pp. 993-1009, 1972.

[38] C. Moretti, J. Bulosan, D. Thain, and P. J. Flynn, "All-Pairs: An Abstraction for Data-Intensive Cloud Computing," in *IEEE International Parallel & Distributed Processing Symposium, IPDPS'08*, pp. 1-11, 2008.

[39] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: A tool for the composition and enactment of bioinformatics workflows," *Bioinformatics,* vol. 20, pp. 3045-3054, 2004.

[40] M. T. Ozsu and P. Valduriez, *Principles of distributed database systems*: Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1991.

[41] R. Prodan and T. Fahringer, "Overhead Analysis of Scientific Workflows in Grid Environments," *IEEE Transactions on Parallel and Distributed Systems,* vol. 19, pp. 378-393, 2008.

[42] G. Singh, K. Vahi, A. Ramakrishnan, G. Mehta, E. Deelman, H. Zhao, R. Sakellariou, K. Blackburn, D. Brown, S. Fairhurst, D. Meyers, G. B. Berriman, J. Good, and D. S. Katz, "Optimizing Workflow Data Footprint," *Scientific Programming,* vol. 15, pp. 249-268, 2007.

[43] S. Venugopal and R. Buyya, "An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids," *J. Parallel Distrib. Comput.,* vol. 68, pp. 471-481, 2008.

[44] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing," *ACM Comput. Surv.,* vol. 38, pp. 1-53, 2006.

[45] S. Venugopal, R. Buyya, and L. Winton, "A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids," in *2nd Workshop on Middleware in Grid Computing*, Toronto, Canada, pp. 75-80, 2004.

[46] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific Cloud Computing: Early Definition and Experience," in *10th IEEE International Conference on High Performance Computing and Communications, HPCC '08.* , pp. 825-830, 2008.

[47] A. Weiss, "Computing in the Cloud," *ACM Networker,* vol. 11, pp. 18-25, 2007.

[48] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of Scientific Workflows in the ASKALON Grid Environment," *SIGMOD Record,* vol. 34, pp. 56-62, 2005.

[49] T. Xie, "SEA: A Striping-Based Energy-Aware Strategy for Data Placement in RAID-Structured Storage Systems," *IEEE Transactions on Computers,* vol. 57, pp. 748-761, 2008.

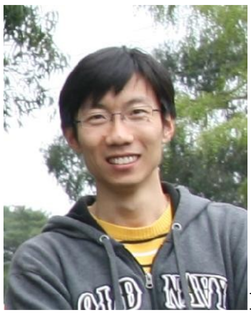
[50] J. Yan, Y. Yang, and G. K. Raikundalia, "SwinDeW - A P2P-Based Decentralized Workflow Management System," *IEEE Transactions on Systems, Man and Cybernetics, Part A,* vol. 36, pp. 922-935, 2006.

[51] Y. Yang, K. Liu, J. Chen, J. Lignier, and H. Jin, "Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G," in *IEEE International Conference on e-Science and Grid Computing*, pp. 51-58, 2007.

[52] Y. Yang, K. Liu, J. Chen, X. Liu, D. Yuan, and H. Jin, "An Algorithm in SwinDeW-C for Scheduling Transaction-Intensive Cost-Constrained Cloud Workflows," in *4th IEEE International Conference on e-Science*, pp. 374-375, 2008.

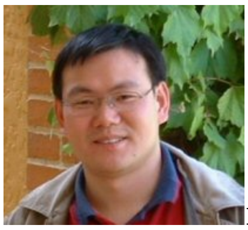
DENOTATIONS:

d_i	dataset
D	set of datasets
D_i	set of datasets in a partition
fd_i	fixed location dataset
FD	set of fixed location datasets
t_i	workflow task
T	set of workflow tasks
T_i	set of workflow tasks that will use dataset d_i
dc_i	data centre
DC	set of data centres
p_i	partition of datasets
P	set of partitions
s_i	size of a dataset
cs	size of a data centre
ds	size of a partition
ps	size of a set of partitions
FP	set of partitions that have fixed location datasets
NFP	set of partitions that do not have fixed location datasets
DM	dependency matrix
CM	clustered dependency matrix
CM_i	sub clustered dependency matrix
CM_T	the top sub clustered dependency matrix after one binary partition
CM_B	the bottom sub clustered dependency matrix after one binary partition
GM	global measure of BEA transformation
PM	global measure of binary partition
dep_{ij}	dependency between datasets d_i and d_j
dc_dep_{ij}	dependency between dataset d_i and data centre dc_j
K	set of data centres with placement of datasets
λ_{ini}	initial storage usage parameter of data centres
λ_{max}	maximum storage usage parameter of data centres


Dong Yuan was born in Jinan, China. He received the B.Eng. degree in 2005 and M.Eng. degree in 2008 both from Shandong University, Jinan, China, all in computer science.
He is currently a PhD student in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Vic., Australia. His research interests include data management in workflow systems, scheduling and resource management, grid and cloud computing.

Yun Yang was born in Shanghai, China. He received the B.S. degree from Anhui University, Hefei, China, in 1984, the M.Eng. degree from the University of Science and Technology of China, Hefei, China, in 1987, and the Ph.D. degree from the University of Queensland, Brisbane, Australia, in 1992, all in computer science.
He is currently a Full Professor in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Vic., Australia. Prior to joining Swinburne as an Associate Professor, he was a Lecturer and Senior Lecturer at Deakin University during 1996-1999. Before that, he was a (Senior) Research Scientist at DSTC Cooperative Research Centre for Distributed Systems Technology during 1993-1996. He also worked at the Beijing University of Aeronautics and Astronautics during 1987-1988. He has co-edited two books and published more than 170 papers on journals and refereed conferences. His current research interests include software technologies, p2p/grid/cloud workflow systems, service-oriented computing, cloud computing, and e-learning.

Xiao Liu received his master degree in management science and engineering from Hefei University of Technology, Hefei, China, 2007. He is currently a PhD student in Centre for Complex Software Systems and Services in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. His research interests include workflow management systems, scientific workflow, business process management and data mining.



Dr. Jinjun Chen received his Ph.D. degree from Swinburne University of Technology, Australia. His thesis was granted Research Thesis Exececllence Award. He received Swinburne Vice Chancellor's research award 2008. He is a core executive member of IEEE Technicial Committee of Scalable Computing and the coordinator of IEEE TCSC technicial area of Workflow Management in Scalable Computing Environments. He is the Editor-in-Chief of Springer book series on Advances in Business Process and Workflow Management (http://www.swinflow.org/books/springer/SpringerBook.htm) and Editor-in-Chief of Nova book series on Process and Workflow Management and Applications(http://www.swinflow.org/books/nova/NovaBook.htm). He has guest edited or is editing several special issues in quality journals such as in IEEE Transactions on Automation Science and Engineering. He has been involved in the organization of many conferences and awarded IEEE Computer Society Service Award (2007).
He has published more than 50 papers in journals and conferences such as ICSE2008 and ACM TAAS. His research interests include Scientific Workflow Management and Applications, Workflow Management and Applications in Web Service or SOC Environments, Workflow Management and Applications in Grid (Service)/Cloud Computing Environments, Software Verification and Validation in Workflow Systems, QoS and Resource Scheduling in Distributed Computing Systems such as Cloud Computing, Service Oriented Computing (SLA and Composition).