

A Cost-Effective Strategy for Intermediate Data Storage in Scientific Cloud Workflow Systems

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—Many scientific workflows are data intensive where a large volume of intermediate data is generated during their execution. Some valuable intermediate data need to be stored for sharing or reuse. Traditionally, they are selectively stored according to the system storage capacity, determined manually. As doing science on cloud has become popular nowadays, more intermediate data can be stored in scientific cloud workflows based on a pay-for-use model. In this paper, we build an Intermediate data Dependency Graph (IDG) from the data provenances in scientific workflows. Based on the IDG, we develop a novel intermediate data storage strategy that can reduce the cost of the scientific cloud workflow system by automatically storing the most appropriate intermediate datasets in the cloud storage. We utilise Amazon’s cost model and apply the strategy to an astrophysics pulsar searching scientific workflow for evaluation. The results show that our strategy can reduce the overall cost of scientific cloud workflow execution significantly.

Keywords - data storage; cost; scientific workflow; cloud computing

I. INTRODUCTION

Scientific applications are usually complex and data-intensive. In many fields, like astronomy [9], high-energy physics [17] and bio-informatics [19], scientists need to analyse terabytes of data either from existing data resources or collected from physical devices. The scientific analyses are usually computation intensive, hence taking a long time for execution. Workflow technologies can be facilitated to automate these scientific applications. Accordingly, scientific workflows are typically very complex. They usually have a large number of tasks and need a long time for execution. During the execution, a large volume of new intermediate data will be generated [10]. They could be even larger than the original data and contain some important intermediate results. After the execution of a scientific workflow, some intermediate data may need to be stored for future use because: 1) scientists may need to re-analyse the results or apply new analyses on the intermediate data; 2) for collaboration, the intermediate results are shared among scientists from different institutions and the intermediate data can be reused. Storing valuable intermediate data can save their regeneration cost when they are reused, not to mention the waiting time saved for regeneration. Given the large size of the data, running scientific workflow applications usually

need not only high performance computing resources but also massive storage [10].

Nowadays, popular scientific workflows are often deployed in grid systems [17] because they have high performance and massive storage. However, building a grid system is extremely expensive and it is normally not open for scientists all over the world. The emergence of cloud computing technologies offers a new way to develop scientific workflow systems in which one research topic is cost-effective strategies for storing intermediate data.

In late 2007 the concept of cloud computing was proposed [23] and it is deemed as the next generation of IT platforms that can deliver computing as a kind of utility [8]. Foster et al. made a comprehensive comparison of grid computing and cloud computing [12]. Cloud computing systems provide the 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 [23]. Research into doing science and data-intensive applications on the cloud has already commenced [18], such as early experiences like Nimbus [15] and Cumulus [22] projects. The work by Deelman et al. [11] shows that cloud computing offers a cost-effective solution for data-intensive applications, such as scientific workflows [14]. Furthermore, cloud computing systems offer a new model 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 in the cloud. Scientists can upload their data and launch their applications on the scientific cloud workflow systems from everywhere in the world via the Internet, and they only need to pay for the resources that they use for their applications. As all the data are managed in the cloud, it is easy to share data among scientists.

Scientific cloud workflows are deployed in a cloud computing environment, where all the resources need to be paid for use. For a scientific cloud workflow system, storing all the intermediated data generated during workflow executions may cause a high storage cost. On the contrary, if we delete all the intermediate data and regenerate them every time when needed, the computation

cost of the system may well be very high too. The intermediate data management is to reduce the total cost of the whole system. The best way is to find a balance that selectively store some popular datasets and regenerate the rest of them when needed.

In this paper, we propose a novel strategy for the intermediate data storage of scientific cloud workflows to reduce the overall cost of the system. The intermediate data in scientific cloud workflows often have dependencies. Along workflow execution, they are generated by the tasks. A task can operate on one or more datasets and generate new one(s). These generation relationships are a kind of data provenance. Based on the data provenance, we create an Intermediate data Dependency Graph (IDG), which records the information of all the intermediate datasets that have ever existed in the cloud workflow system, no matter whether they are stored or deleted. With the IDG, the system knows how the intermediate datasets are generated and can further calculate their generation cost. Given an intermediate dataset, we divide its generation cost by its usage rate, so that this cost (the generation cost per unit time) can be compared with its storage cost per time unit, where a dataset's usage rate is the time between every usage of this dataset that can be obtained from the system log. Our strategy can automatically decide whether an intermediate dataset should be stored or deleted in the cloud system by

comparing the generation cost and storage cost, and no matter this intermediate dataset is a new dataset, regenerated dataset or stored dataset in the system.

The reminder of this paper is organised as follows. Section 2 gives a motivating example and analyses the research problems. Section 3 introduces some important related concepts to our strategy. Section 4 presents the detailed algorithms in our strategy. Section 5 demonstrates the simulation results and the evaluation. Section 6 discusses the related work. Section 7 addresses our conclusions and future work.

II. MOTIVATING EXAMPLE AND PROBLEM ANALYSIS

2.1 Motivating example

Scientific applications often need to process a large amount of data. For example, Swinburne Astrophysics group has been conducting a pulsar searching survey using the observation data from Parkes Radio Telescope (<http://astronomy.swin.edu.au/pulsar/>), which is one of the most famous radio telescopes in the world (<http://www.parkes.atnf.csiro.au>). Pulsar searching is a typical scientific application. It contains complex and time consuming tasks and needs to process terabytes of data. Fig. 1 depicts the high level structure of a pulsar searching workflow.

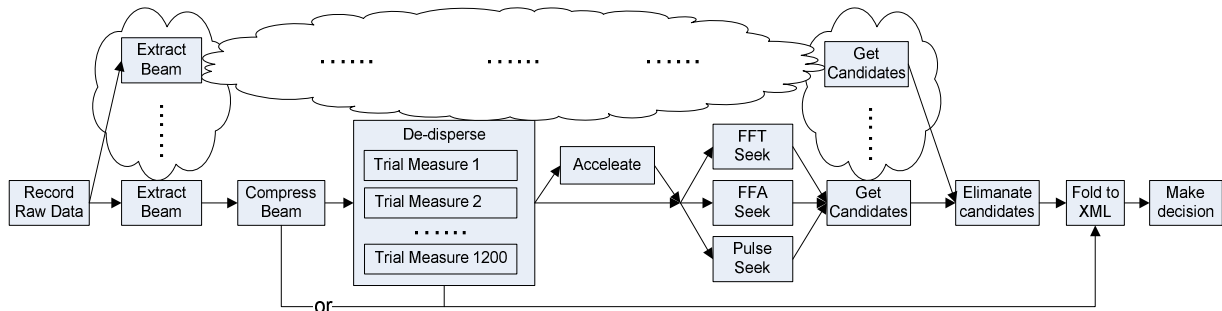


Figure 1. Pulsar searching workflow

At the beginning, raw signal data from Parkes Radio Telescope are recorded at a rate of one gigabyte per second by the ATNF¹ Parkes Swinburne Recorder (APSR). Depends on different areas in the universe that the researchers want to conduct the pulsar searching survey, the observation time is normally from 4 minutes to one hour. Recording from the telescope in real time, these raw data files have data from multiple beams interleaved. For initial preparation, different beam files are extracted from the raw data files and compressed. They are 1GB to 20GB each in size, depends on the observation time. The beam files contain the pulsar signals which are dispersed by the interstellar medium. De-dispersion is to counteract this effect. Since the potential dispersion source is unknown, a large number of de-dispersion files will be generated with different dispersion trials. In the current pulsar searching

survey, 1200 is the minimum number of the dispersion trials. Based on the size of the input beam file, this de-dispersion step will take 1 to 13 hours to finish and generate up to 90GB of de-dispersion files. Furthermore, for binary pulsar searching, every de-dispersion file will need another step of processing named accelerate. This step will generate the accelerated de-dispersion files with the similar size in the last de-dispersion step. Based on the generated de-dispersion files, different seeking algorithms can be applied to search pulsar candidates, such as FFT Seeking, FFA Seeking, and Single Pulse Seeking. For a large input beam file, it will take more than one hour to seek the 1200 de-dispersion files. A candidate list of pulsars will be generated after the seeking step which is saved in a text file. Furthermore, by comparing the candidates generated from different beam files in a same time session, some interference may be detected and some candidates may be eliminated. With the final pulsar candidates, we need to go back to the beam files or the de-

¹ ATNF refers to the Australian Telescope National Facility.

dispersion files to find their feature signals and fold them to XML files. At last, the XML files will be visual displayed to researchers for making decisions on whether a pulsar has been found or not.

As described above, we can see that this pulsar searching workflow is both computation and data intensive. It is currently running on Swinburne high performance supercomputing facility (<http://astronomy.swinburne.edu.au/supercomputing/>). It needs long execution time and a large amount of intermediate data is generated. At present, all the intermediate data are deleted after having been used, and the scientists only store the raw beam data, which are extracted from the raw telescope data. Whenever there are needs of using the intermediate data, the scientists will regenerate them based on the raw beam files. The intermediated data are not stored, mainly because the supercomputer is a shared facility that can not offer unlimited storage capacity to hold the accumulated terabytes of data. However, some intermediate data are better to be stored. For example, the de-dispersion files are frequently used intermediate data. Based on them, the scientists can apply different seeking algorithms to find potential pulsar candidates. Furthermore, some intermediate data are derived from the de-dispersion files, such as the results of the seek algorithms and the pulsar candidate list. If these data are reused, the de-dispersion files will also need to be regenerated. For the large input beam files, the regeneration of the de-dispersion files will take more than 10 hours. It not only delays the scientists from conducting their experiments, but also wastes a lot of computation resources. On the other hand, some intermediate data may not need to be stored. For example, the accelerated de-dispersion files, which are generated by the accelerate step. The accelerate step is an optional step that is only for the binary pulsar searching. Not all pulsar searching processes need to accelerate the de-dispersion files, so the accelerated de-dispersion files are not that often used. In light of this and given the large size of these data, they are not worth to store as it would be more cost effective to regenerate them from the de-dispersion files whenever they are used.

2.2 Problem analysis

Traditionally, scientific workflows are deployed on the high performance computing facilities, such as clusters and grids. Scientific workflows are often complex with huge intermediate data generated during their execution. How to store these intermediate data is normally decided by the scientists who use the scientific workflows. This is because the clusters and grids only serve for certain institutions. The scientists may store the intermediate data that are most valuable to them, based on the storage capacity of the system. However, in many scientific workflow systems, the storage capacities are limited, such as the pulsar searching workflow we introduced. The scientists have to delete all the intermediate data because of the storage limitation. This bottleneck of storage can be avoided if we run scientific workflows in the cloud.

In a cloud computing environment, theoretically, the system can offer unlimited storage resources. All the intermediate data generated by scientific cloud workflows can be stored, if we are willing to pay for the required resources. However, in scientific cloud workflow systems, whether to store intermediate data or not is not an easy decision anymore.

1) All the resources in the cloud carry certain costs, so either storing or generating an intermediate dataset, we have to pay for the resources used. The intermediate datasets vary in size, and have different generation cost and usage rate. Some of them may often be used whilst some others may be not. On one hand, it is most likely not cost effective to store all the intermediate data in the cloud. On the other hand, if we delete them all, regeneration of frequently used intermediate datasets imposes a high computation cost. We need a strategy to balance the generation cost and the storage cost of the intermediate data, in order to reduce the total cost of the scientific cloud workflow system. In this paper, given the large capacity of data centre and the consideration of cost effectiveness, we assuming that all the intermediate data are stored in one data centre, therefore, data transfer cost is not considered.

2) The scientists can not predict the usage rate of the intermediate data anymore. For a single research group, if the data resources of the applications are only used by its own scientists, the scientists may predict the usage rate of the intermediate data and decide whether to store or delete them. However, the scientific cloud workflow system is not developed for a single scientist or institution, rather, developed for scientists from different institutions to collaborate and share data resources. The users of the system could be anonymous from the Internet. We must have a strategy storing the intermediate data based on the needs of all the users that can reduce the cost of the whole system.

Hence, for scientific cloud workflow systems, we need a strategy that can automatically select and store the most appropriate intermediate datasets. Furthermore, this strategy should be cost effective that can reduce the total cost of the whole system.

III. COST ORIENTED INTERMEDIATE DATA STORAGE IN SCIENTIFIC CLOUD WORKFLOWS

3.1 Data management in scientific cloud workflow systems

In a cloud computing system, application data are stored in large data centres. The cloud users visit the system via the Internet and upload the data to conduct their applications. All the application data are stored in the cloud storage and managed by the cloud system independent of users. As time goes on and the number of cloud users increases, the volume of data stored in cloud will become huge. This makes the data management in cloud computing system a very challenging job.

Scientific cloud workflow system is the workflow system for scientists to run their applications in the cloud.

As depicted in Figure 2, it has many differences with the traditional scientific workflow systems in data

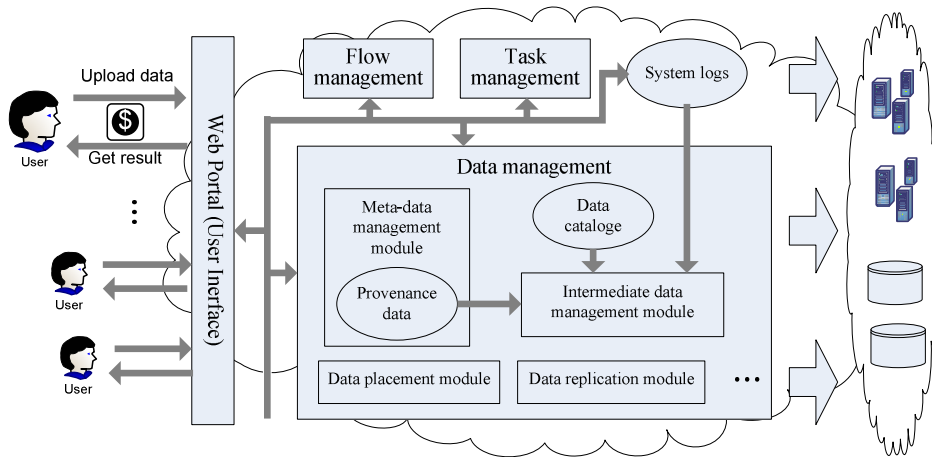


Figure 2. Structure of data management in scientific cloud workflow system

management. The most important ones are as follows. 1) For scientific cloud workflows, all the application data are managed in the cloud. To launch their workflows, the scientists have to upload their application data to the cloud storage via a Web portal. This requires data management to be automatic. 2) The scientific cloud workflow system has a cost model. The scientists have to pay for the resources used for conducting their applications. Hence, the data management has to be cost oriented. 3) The scientific cloud workflow system is based on the Internet, where the application data are shared and reused among the scientists world wide. For the data reanalyses and regenerations, data provenance is more important in scientific cloud workflows.

In general, there are two types of data stored in the cloud storage, *input data* and *intermediate data* including result data. First, input data are the data uploaded by users, and in the scientific applications they also can be the raw data collected from the devices. These data are the original data for processing or analysis that are usually the input of the applications. The most important feature of these data is that if they were deleted, they could not be regenerated by the system. Second, intermediate data are the data newly generated in the cloud system while the application runs. These data save the intermediate computation results of the application that will be used in the future execution. In general, the final result data of the applications are a kind of intermediate data, because the result data in one application can also be used in other applications. When further operations apply on the result data, they become intermediate data. Hence, the intermediate data are the data generated based on either the input data or other intermediate data, and the most important feature is that they can be regenerated if we know their provenance.

For the input data, the users will decide whether they should be stored or deleted, since they can not be regenerated once deleted. For the intermediate data, their storage status can be decided by the system, since they can

be regenerated. Hence, in this paper we develop a strategy for intermediate data storage that can significantly reduce the cost of scientific cloud workflow system.

3.2 Data provenance and Intermediate data Dependency Graph (IDG)

Scientific workflows have many computation and data intensive tasks that will generate many intermediate datasets of considerable size. There are dependencies exist among the intermediate datasets. Data provenance in workflows is a kind of important metadata, in which the dependencies between datasets are recorded [21]. The dependency depicts the derivation relationship between workflow intermediate datasets. For scientific workflows, data provenance is especially important, because after the execution, some intermediate datasets may be deleted, but sometimes the scientists have to regenerate them for either reuse or reanalysis [7]. Data provenance records the information of how the intermediate datasets were generated, which is very important for the scientists. Furthermore, regeneration of the intermediate datasets from the input data may be very time consuming, and therefore carry a high cost. With data provenance information, the regeneration of the demanding dataset may start from some stored intermediated datasets instead. In the scientific cloud workflow system, data provenance is recorded while the workflow execution. Taking advantage of data provenance, we can build an IDG based on data provenance. All the intermediate datasets once generated in the system, whether stored or deleted, their references are recorded in the IDG.

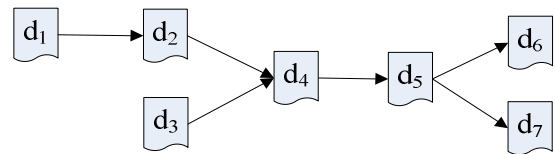


Figure 3. A simple Intermediate data Dependency Graph (IDG)

IDG is a directed acyclic graph, where every node in the graph denotes an intermediate dataset. Figure 3 shows us a simple IDG, dataset d_1 is pointed to d_2 means d_1 is used to generate d_2 ; dataset d_2 and d_3 are pointed to d_4 means d_2 and d_3 are used together to generate d_4 ; and d_5 is pointed to d_6 and d_7 means d_5 is used to generate either d_6 or d_7 based on different operations. In the IDG, all the intermediate datasets' provenances are recorded. When some of the deleted intermediate datasets need to be reused, we do not need to regenerate them from the original input data. With the IDG, the system can find the predecessor datasets of the demanding data, so they can be regenerated from their nearest existing predecessor datasets.

3.3 Cost model

With the IDG, given any intermediate datasets that ever occurred in the system, we know how to regenerate it. However, in this paper, we aim at reducing the total cost of managing the intermediate data. In a cloud computing environment, if the users want to deploy and run applications, they need to pay for the resources used. The resources are offered by cloud service providers, who have their cost models to charge the users. In general, there are two basic types of resources in cloud computing: storage and computation. Popular cloud services providers' cost models are based on these two types of resources [1]. Furthermore, the cost of data transfer is also considered, such as in Amazon's cost model. In [11], the authors state that a cost-effective way of doing science in the cloud is to upload all the application data to the cloud and run all the applications in the cloud services. So we assume that the scientists upload all the input data to the cloud to conduct their experiments. Because transferring data within one cloud service provider's facilities is usually free, the data transfer cost of managing intermediate data during the workflow execution is not counted. In this paper, we define our cost model for managing the intermediate data in a scientific cloud workflow system as follows:

$$Cost=C+S,$$

where the total cost of the system, $Cost$, is the sum of C , which is the total cost of computation resources used to regenerate the intermediate data, and S , which is the total cost of storage resources used to store the intermediate data. For the resources, different cloud service providers have different prices. In this paper, we use Amazon services' price as follows:

\$0.15 per Gigabyte per month for the storage resources.

\$0.1 per CPU hour for the computation resources.

Furthermore, we denote these two prices as $CostS$ and $CostC$ for the algorithms respectively.

To utilise the cost model, we define some important attributes for the intermediate datasets in the IDG. For intermediate dataset d_i , its attributes are denoted as: $\langle size, flag, t_p, t, pSet, fSet, CostR \rangle$, where

$size$, denotes the size of this dataset;

$flag$, denotes the status whether this dataset is stored or deleted in the system;

t_p , denotes the time of generating this dataset from its direct predecessor datasets;

t , denotes the usage rate, which is the time between every usage of d_i in the system. In traditional scientific workflows, t can be defined by the scientists, who use this workflow collaboratively. However, a scientific cloud workflow system is based on the Internet with large number of users, as we discussed before, d_i can not be defined by users. It is a forecasting value from the dataset's usage history recorded in the system logs. t is a dynamic value that changes according to d_i 's real usage rate in the system.

$pSet$, is the set of references of all the deleted intermediate datasets in the IDG that linked to d_i , which is shown in Figure 4. If we want to regenerate d_i , $d_i.pSet$ contains all the datasets that need to be regenerated beforehand. Hence, the generation cost of d_i can be denoted as:

$$genCost(d_i) = (d_i.t_p + \sum_{d_j \in d_i.pSet} d_j.t_p) * CostC;$$

$fSet$, is the set of references of all the deleted intermediate datasets in the IDG that are linked by d_i , which is shown in Figure 4. If d_i is deleted, to regenerate any datasets in $d_i.fSet$, we have to regenerate d_i first. In another word, if the storage status of d_i has changed, the generation cost of all the datasets in $d_i.fSet$ will be affected by $genCost(d_i)$;

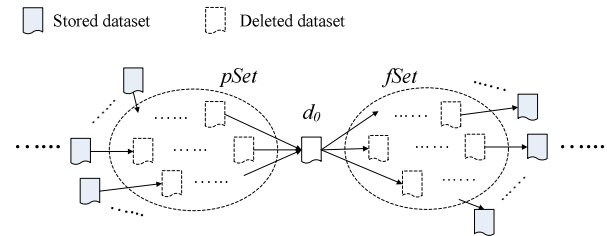


Figure 4. A segment of IDG

$CostR$, is d_i 's cost rate, which means the average cost per time unit of the dataset d_i in the system, in this paper we use hour as time unit. If d_i is a stored dataset, $d_i.CostR = d_i.size * CostS$. If d_i is a deleted dataset in the system, when we need to use d_i , we have to regenerate it. So we divide the generation cost of d_i by the time between its usages and use this value as the cost rate of d_i in the system. $d_i.CostR = genCost(d_i) / d_i.t$. When the storage status of d_i is changed, its $CostR$ will be changed correspondingly.

Hence, the system cost rate of managing intermediate data is the sum of $CostR$ of all the intermediate datasets, which is $\sum_{d_i \in IDG} (d_i.CostR)$. Given time duration, denoted as $[T_0, T_n]$, the total system cost is the integral of the system cost rate in this duration as a function of time t , which is:

$$Total_Cost = \int_{t=T_0}^{T_n} \left(\sum_{d_i \in IDG} (d_i.CostR) \right) \bullet dt$$

The goal of our intermediate data management is to reduce this cost. In the next section, we will introduce a dependency based intermediate data storage strategy, which selectively stores the intermediate datasets to reduce the total cost of the scientific cloud workflow system.

IV. DEPENDENCY BASED INTERMEDIATE DATA STORAGE STRATEGY

The IDG records the references of all the intermediate datasets and their dependencies that ever occurred in the system, some datasets may be stored in the system, and others may be deleted. When new datasets are generated in the system, their information is added to the IDG at the first time. Our dependency based intermediate data storage strategy is developed based on the IDG, and applied at workflow runtime. It can dynamically store the essential intermediate datasets during workflow execution. The strategy contains three algorithms described in this section.

Input: a newly generated intermediate dataset d_0 ;
an IDG ;

Output: storage strategy of d_0 ;

```

add  $d_0$ 's information to IDG ;
 $genCost(d_0) = (\sum_{d_i \in d_0.pSet} d_i.t_p + d_0.t_p) * CostC$ ; //Calculate  $d_0$ 's generation cost
if ( $genCost(d_0)/d_0.t > d_0.size * CostS$ ) //Compare  $d_0$ 's storage and generation cost rate
    {  $d_0.flag = 'stored'$ ; //decide to store  $d_0$ 
       $d_0.CostR = d_0.size * CostS$ ; //set the cost rate
    }
else
    {  $d_0.flag = 'deleted'$ ; //decide to delete  $d_0$ 
       $d_0.CostR = genCost(d_0)/d_0.t$ ; //set the cost rate
    }
update IDG & execute 'store' or 'delete' on  $d_0$ ;

```

Figure 5. Algorithm for handling newly generated datasets

In this algorithm, we guarantee that all the intermediate datasets chosen to be stored are necessary, which means that deleting anyone of them would increase the cost to the system, since they all have a higher generation cost than storage cost.

4.2 Algorithm for managing stored intermediate datasets

The usage rate t of a dataset is an important parameter that determines its storage status. Since t is a dynamic value that may change at any time, we have to dynamically check the stored intermediate datasets in the system that whether they still need to be stored.

For an intermediate dataset d_0 that is stored in the system, we set a threshold time t_θ , where $d_0.t_\theta = genCost(d_0)/(d_0.size * CostS)$. This threshold time indicates how long this dataset can be stored in the system with the cost of generating it. If d_0 has not been used for the time of t_θ , we will check whether it should be stored anymore.

If we delete stored intermediate dataset d_0 , the system cost rate is reduced by d_0 's storage cost rate, which is

4.1 Algorithm for deciding newly generated intermediate datasets' storage status

Suppose d_0 is a newly generated intermediate dataset.

First, we add its information to the IDG. We find the provenance datasets of d_0 in the IDG, and add edges pointed to d_0 from these datasets. Then we initialise its attributes. As d_0 does not have a usage history yet, we use the average value in the system as the initial value of d_0 's usage rate.

Next, we check if d_0 needs to be stored or not. As d_0 is newly added in the IDG, it does not have successor datasets in the IDG, which means no intermediate datasets are derived from d_0 at this moment. For deciding whether to store or delete d_0 , we only compare the generation cost and storage cost of d_0 itself, which are $genCost(d_0)/d_0.t$ and $d_0.size * CostS$. If the cost of generation is larger than the cost of storing it, we save d_0 and set $d_0.CostR = d_0.size * CostS$, otherwise we delete d_0 and set $d_0.CostR = genCost(d_0)/d_0.t$. The algorithm is shown in Figure 5.

$d_0.size * CostS$. Meanwhile, the increase of the system cost rate is the sum of the generation cost rate of d_0 itself, which is $genCost(d_0)/d_0.t$, and the increased generation cost rates of all the datasets in $d_0.fSet$ caused by deleting d_0 , which is $\sum_{d_i \in d_0.fSet} (genCost(d_0)/d_i.t)$. We compare d_0 's storage cost rate and generation cost rate to decide whether d_0 should be stored or not. The detailed algorithm is shown in Figure 6.

Lemma: The deletion of stored intermediate dataset d_0 in the IDG does not affect the stored datasets adjacent to d_0 , where the stored datasets adjacent to d_0 means the datasets that directly link to d_0 or $d_0.pSet$, and the datasets that are directly linked by d_0 or $d_0.fSet$.

Proof:

1) Suppose d_p is a stored dataset directly linked to d_0 or $d_0.pSet$. Since d_0 is deleted, d_0 and $d_0.fSet$ are added to $d_p.fSet$. So the new generation cost rate of d_p in the system is $genCost(d_p)/d_p.t + \sum_{d_i \in d_p.fSet \cup d_0 \cup d_0.fSet} (genCost(d_p)/d_i.t)$, and it is larger than before, which was

$genCost(d_p)/d_p.t + \sum_{d_i \in d_p.fSet} (genCost(d_p)/d_i.t)$. Hence d_p still needs to be stored;

2) Suppose d_f is a stored dataset directly linked by d_0 or $d_0.fSet$. Since d_0 is deleted, d_0 and $d_0.pSet$ are added to $d_f.pSet$. So the new generation cost of d_f is $genCost(d_f) = (d_f.t_p + \sum_{d_i \in d_f.pSet \cup d_0 \cup d_0.pSet} d_i.t_p) * CostC$, and it is larger than before, which was

$genCost(d_f) = (d_f.t_p + \sum_{d_i \in d_f.pSet} d_i.t_p) * CostC$. Because of the increase of $genCost(d_f)$, the generation cost rate of d_f in the system is larger than before, which was $genCost(d_f)/d_f.t + \sum_{d_i \in d_f.fSet} (genCost(d_f)/d_i.t)$. Hence d_f still needs to be stored.

Because of 1) and 2), the Lemma holds.

Input: a stored intermediate dataset d_0 ;
an IDG ;
Output: storage strategy of d_0 ;

```

genCost(d_0) = (sum_{d_i in d_0.pSet} d_i.t_p + d_0.t_p) * CostC; //calculate d_0's generation cost
if (genCost(d_0)/d_0.t + sum_{d_i in d_0.fSet} (genCost(d_0)/d_i.t) > d_0.size * CostS) //compare d_0's storage and generation cost rate
    T' = T + d_0.t_theta; //set the next checking time T', T is the current system time, t_theta is the duration d_0 should be stored
else
    {
        d_0.flag = 'deleted'; //decide to delete d_0
        d_0.CostR = genCost(d_0)/d_0.t; //change d_0's cost rate
        for (every d_i in d_0.fSet) //change the cost rates of all the datasets in d_0.fSet
            d_i.CostR = d_i.CostR + genCost(d_0)/d_i.t; //cost rate increases with the generation cost of d_0
    }
update IDG & execute 'store' or 'delete' of d_0;

```

Figure 6. Algorithm for checking stored intermediate datasets

By applying the algorithm of checking the stored intermediate datasets, we can still guarantee that all the datasets we have kept in the system are necessary to be stored. Furthermore, when the deleted intermediate datasets are regenerated, we also need to check whether to store or delete them as discussed next.

4.3 Algorithm for deciding the regenerated intermediate datasets' storage status

The IDG is a dynamic graph where the information of new intermediate datasets may join at anytime. Although the algorithms in the above two sub-sections can guarantee that the stored intermediate datasets are all necessary, these stored datasets may not be the most cost effective. Initially deleted intermediate datasets may need to be stored as the IDG expands. Suppose d_0 is a regenerated intermediate dataset in the system, which has been deleted before. After been used, we have to recalculate d_0 's storage status, as well as the stored datasets adjacent to d_0 in the IDG.

Theorem: If regenerated intermediate dataset d_0 is stored, only the stored datasets adjacent to d_0 in the IDG may need to be deleted to reduce the system cost.

Proof:

1) Suppose d_p is a stored dataset directly linked to d_0 or $d_0.pSet$. Since d_0 is stored, d_0 and $d_0.fSet$ need to be removed from $d_p.fSet$. So the new generation cost rate of d_p in the system is $genCost(d_p)/d_p.t + \sum_{d_i \in d_p.fSet - d_0 - d_0.fSet} (genCost(d_p)/d_i.t)$, and it is smaller than before, which was $genCost(d_p)/d_p.t + \sum_{d_i \in d_p.fSet} (genCost(d_p)/d_i.t)$. If the new generation cost rate is smaller than the storage cost rate of d_p , d_p would be deleted. The rest of the stored

intermediate datasets are not affected by the deletion of d_p , because of the Lemma introduced before.

2) Suppose d_f is a stored dataset directly linked by d_0 or $d_0.fSet$. Since d_0 is stored, d_0 and $d_0.pSet$ need to be removed from $d_f.pSet$. So the new generation cost of d_f is $genCost(d_f) = (d_f.t_p + \sum_{d_i \in d_f.pSet - d_0 - d_0.pSet} d_i.t_p) * CostC$, and it is smaller than before, which was $genCost(d_f) = (d_f.t_p + \sum_{d_i \in d_f.pSet} d_i.t_p) * CostC$. Because of the reduce of $genCost(d_f)$, the generation cost rate of d_f in the system is smaller than before, which was $genCost(d_f)/d_f.t + \sum_{d_i \in d_f.fSet} (genCost(d_f)/d_i.t)$. If the new generation cost rate is smaller than the storage cost rate of d_f , d_f would be deleted. The rest of the stored intermediate datasets are not affected by the deletion of d_f , because of the Lemma introduced before.

Because of 1) and 2), the Theorem holds.

If we store regenerated intermediate dataset d_0 , the cost rate of the system increases with d_0 's storage cost rate, which is $d_0.size * CostS$. Meanwhile, the reduction of the system cost rate may be resulted from three aspects:

- (1) The generation cost rate of d_0 itself, which is $genCost(d_0)/d_0.t$;
- (2) The reduced generation cost rates of all the datasets in $d_0.fSet$ caused by storing d_0 , which is $\sum_{d_i \in d_0.fSet} (genCost(d_0)/d_i.t)$;
- (3) As indicated in the Theorem, some stored datasets adjacent to d_0 may be deleted that reduces the cost to the system.

We will compare the increase and reduction of the system cost rate to decide whether d_0 should be stored or

not. The detailed algorithm is shown in Figure 7.

Input: a generated intermediate dataset d_0 ;
an IDG ;

Output: storage strategy of d_0 and d_0 's adjacent stored datasets ;

```

 $d_0.flag = 'stored'$ ; //assume  $d_0$  is stored
 $d_0.costR = d_0.size * costS$ ; //change the cost rate of  $d_0$ 
 $\Delta = genCost(d_0)/d_0.t + \sum_{d_i \in d_0.fSet} (genCost(d_0)/d_i.t) - d_0.size * CostS$ ; //the change of system cost rate, if storing  $d_0$ 
for (every  $d_i$  in  $d_0.fSet$ ) //change the cost rates of all the datasets in  $d_0.fSet$ 
     $d_i.costR = d_i.costR - genCost(d_0)/d_i.t$ ; //cost rate increases with the generation cost of  $d_0$ 
for (every  $d_j$  directly linked by  $d_0.fSet$ ) //check if the stored predecessor datasets adjacent to  $d_0$  need to be deleted
    if ( $genCost(d_j)/d_j.t + \sum_{d_m \in d_j.fSet} (genCost(d_j)/d_m.t) < d_j.size * CostS$ ) //compare generation and storage cost rates
         $d_j.flag = 'deleted'$ ; //decide to delete  $d_j$ 
         $d_j.costR = genCost(d_j)/d_j.t$ ; //change the cost rate of  $d_j$ 
        for (every  $d_m$  in  $d_j.fSet$ ) //change the cost rates of all the datasets in  $d_j.fSet$ 
             $d_m.costR = d_m.costR + genCost(d_j)/d_m.t$ ; //cost rate increases with the generation cost of  $d_j$ 
             $\Delta^+ = \Delta^+ + d_j.size * CostS - genCost(d_j)/d_j.t - \sum_{d_m \in d_j.fSet} (genCost(d_j)/d_m.t)$ ; //accumulate cost rate benefit of deleting  $d_j$ 
        for (every  $d_k$  directly linked by  $d_0.pSet$ ) //check if the stored successor datasets adjacent to  $d_0$  need to be deleted
            if ( $genCost(d_k)/d_k.t + \sum_{d_n \in d_k.fSet} (genCost(d_k)/d_n.t) < d_k.size * CostS$ ) //compare generation and storage cost rates
                 $d_k.flag = 'deleted'$ ; //decide to delete  $d_k$ 
                 $d_k.costR = genCost(d_k)/d_k.t$ ; //change the cost rate of  $d_k$ 
                for (every  $d_n$  in  $d_k.fSet$ ) //change the cost rates of all the datasets in  $d_k.fSet$ 
                     $d_n.costR = d_n.costR + genCost(d_k)/d_n.t$ ; //cost rate increases with the generation cost of  $d_k$ 
                     $\Delta^- = \Delta^- + d_k.size * CostS - genCost(d_k)/d_k.t - \sum_{d_n \in d_k.fSet} (genCost(d_k)/d_n.t)$ ; //accumulate cost rate benefit of deleting  $d_k$ 
            if ( $\Delta + \Delta^+ + \Delta^- > 0$ ) //check the change of total system cost rate, if storing  $d_0$  could reduce the system cost
                update IDG and execute 'store' or 'delete' on  $d_0$  and  $d_0$ 's adjacent datasets ;

```

Figure 7. Algorithm for checking deleted intermediate datasets

By applying the algorithm of checking the regenerated intermediate datasets, we can not only guarantee that all the datasets we have kept in the system are necessary to be stored, but also any changes of the datasets' storage status will reduce the total system cost.

V. EVALUATION

5.1 Simulation environment and strategies

The intermediate data storage strategy we proposed in this paper is generic. It can be used in any scientific workflow applications. In this section, we deploy it to the pulsar searching workflow described in Section 2. We use the real world statistics to conduct our simulation on Swinburne high performance supercomputing facility and demonstrate how our strategy works in storing the intermediate datasets of the pulsar searching workflow. To simulate the cloud computing environment, we set up VMware software (<http://www.vmware.com/>) on the physical servers and create virtual clusters as the data centre. Furthermore, we set up the Hadoop file system (<http://hadoop.apache.org/>) in the data centre to manage the application data.

In the pulsar example, during the workflow execution, six intermediate datasets are generated. The IDG of this

pulsar searching workflow is shown in Figure 8, as well as the sizes and generation times of these intermediate datasets. The generation times of the datasets are from running this workflow on Swinburne Supercomputer, and for simulation, we assume that in the cloud system, the generation times of these intermediate datasets are the same. Furthermore, we assume that the prices of cloud services follow Amazon's cost model, i.e. \$0.1 per CPU hour for computation and \$0.15 per gigabyte per month for storage.

To evaluate the performance of our strategy, we run five simulation strategies together and compare the total cost of the system. The strategies are:

- 1) Store all the intermediate datasets in the system;
- 2) Delete all the intermediate datasets, and regenerate them whenever needed;
- 3) Store the datasets that have high generation cost;
- 4) Store the datasets that are most often used; and
- 5) Our strategy to dynamically decide whether a dataset should be stored or deleted.

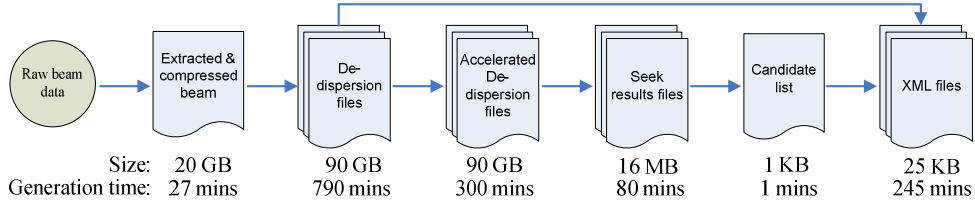


Figure 8. IDG of pulsar searching workflow

5.2 Simulation results

We run the simulations based on the estimated usage rate of every intermediate dataset. From Swinburne astrophysics research group, we understand that the “de-dispersion files” are the most useful intermediate dataset. Based on these files, many accelerating and seeking methods can be used to search pulsar candidates. Hence,

we set the “de-dispersion files” to be used once every 4 days, and rest of the intermediate datasets to be used once every 10 days. Based on this setting, we run the above mentioned five simulation strategies and calculate the total costs of the system for ONE branch of the pulsar searching workflow of processing ONE piece of observation data in 50 days which is shown in Figure 9.

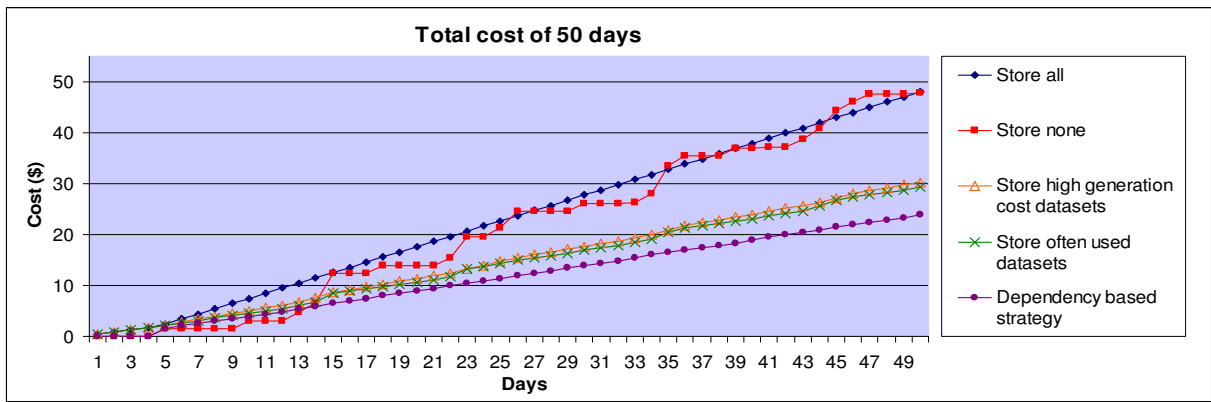


Figure 9. Total cost of pulsar searching workflow with Amazon’s cost model

From Figure 9 we can see that:

- 1) The cost of the “store all” strategy is a straight line, because in this strategy, all the intermediate datasets are stored in the cloud storage that is charged at a fixed rate, and there is no computation cost required;
- 2) The cost of the “store none” strategy is a fluctuated line because in this strategy all the costs are computation cost of regenerating intermediate datasets. For the days that have fewer requests of the data, the cost is low, otherwise, the cost is high;
- 3-5) For the remaining three strategies, the cost lines are only a little fluctuated and the cost is much lower than

the “store all” and “store none” strategies. This is because the intermediate datasets are partially stored.

As indicated in Figure 9 we can draw the conclusion that:

- 1) Neither storing all the intermediate datasets nor deleting them all is a cost-effective way for intermediate data storage;
- 2) Our dependency based strategy performs the most cost effective to store the intermediate datasets.

Furthermore, back to the pulsar searching workflow example, Table 1 shows how the five strategies store the intermediate datasets in detail.

TABLE 1. PULSAR SEARCHING WORKFLOW’S INTERMEDIATE DATASETS STORAGE STATUS IN 5 STRATEGIES

Strategies	Datasets	Extracted beam	De-dispersion files	Accelerated de-dispersion files	Seek results	Pulsar candidates	XML files
Store all		Stored	Stored	Stored	Stored	Stored	Stored
Store none		Deleted	Deleted	Deleted	Deleted	Deleted	Deleted
Store high generation cost datasets		Deleted	Stored	Stored	Deleted	Deleted	Stored
Store often used datasets		Deleted	Stored	Deleted	Deleted	Deleted	Deleted
Dependency based strategy		Deleted	Stored (was deleted initially)	Deleted	Stored	Deleted	Stored

Since the intermediate datasets of this pulsar searching workflow is not complicate, we can do some straightforward analyses on how to store them. For the accelerated de-dispersion files, although its generation cost is quite high, comparing to its huge size, it is not worth to store them in the cloud. However, in the strategy of “store high generation cost datasets”, the accelerated de-dispersion files are chosen to be stored. Furthermore, for the final XML files, they are not very often used, but comparing to the high generation cost and small size, they should be stored. However, in the strategy of “store often used datasets”, these files are not chosen to be stored. Generally speaking, our dependency based strategy is the most appropriate strategy for the intermediate data storage which is also dynamic. From Table 1 we can see, our strategy did not store the de-dispersion files at beginning, but stored them after their regeneration. In our strategy, every storage status change of the datasets would reduce

the total system cost rate, where the strategy can gradually close the minimum cost of system.

One important factor that affects our dependency based strategy is the usage rate of the intermediate datasets. In a system, if the usage rate of the intermediate datasets is very high, the generation cost of the datasets is very high, correspondingly these intermediate datasets are more tend to be stored. On the contrary, in a very low intermediate datasets usage rate system, all the datasets are tend to be deleted. In Figure 9’s simulation, we set the datasets’ usage rate on the borderline that makes the total cost equivalent to the strategies of “store all” and “store none”. Under this condition, the intermediate datasets have no tendency to be stored or deleted, which can objectively demonstrate our strategy’s effectiveness on reducing the system cost. Next we will also demonstrate the performance of our strategy in the situations under different usage rates of the intermediate datasets.

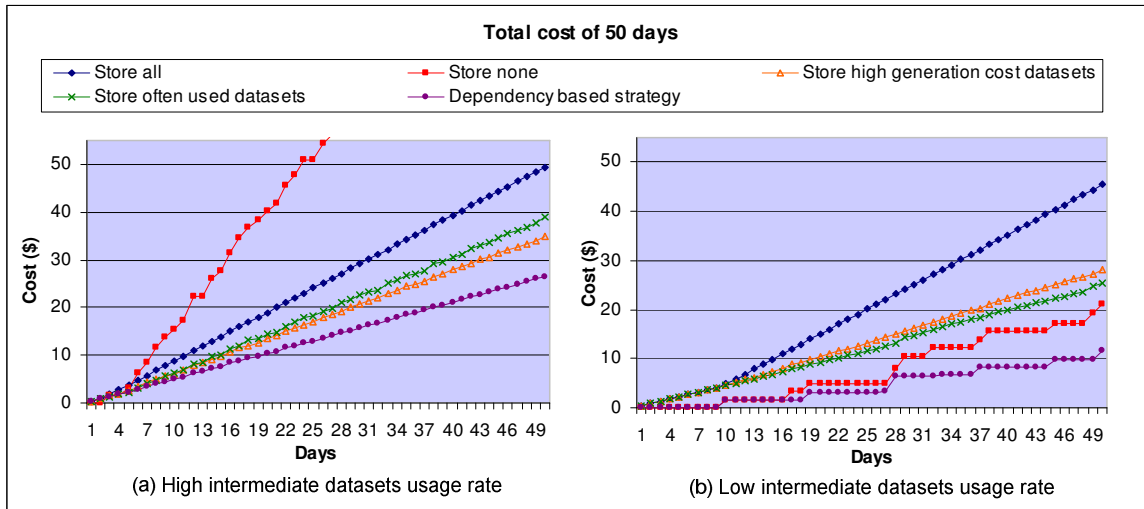


Figure 10. Cost of pulsar searching workflow with different intermediate datasets usage rates

Figure 10 (a) shows the cost of the system with the usage rate of every dataset doubled in the pulsar workflow. From the figure we can see, when the datasets’ usage rates are high, the strategy of “store none” becomes highly cost ineffective, because the frequent regeneration of the intermediate datasets causes a very high cost to the system. In contrast, our strategy is still the most cost-effective one that the total system cost only increases slightly. It is not very much influenced by the datasets’ usage rates. For the “store all” strategy, although it is not influenced by the usage rate, its cost is still very high. The rest two strategies are in the mid range. They are influenced by the datasets’ usage rates more, and their total costs are higher than our strategy.

Figure 10 (b) shows the cost of the system with the usage rate of every dataset halved in the pulsar workflow. From this figure we can see, in the system with a low intermediate datasets reuse rate, the “store all” strategy becomes highly cost ineffective, and the “store none” strategy becomes relatively cost effective. Again, our

strategy is still the most cost-effective one among the five strategies.

From all the simulations we have done on the pulsar searching workflow, we find that depends on different intermediate datasets usage rates, our strategy can reduce the system cost by 46.3%-74.7% in comparison to the “store all” strategy; 45.2%-76.3% to the “store none” strategy; 23.9%-58.9% to the “store high generation cost datasets” strategy; and 32.2%-54.7% “store often used datasets” strategy respectively. Furthermore, to examine the generality of our strategy, we have also conducted many simulations on randomly generated workflows and intermediate data. Due to the space limit, we can not present them here.

Based on the simulations, we can reach the conclusion that our intermediate data storage strategy has a good performance. By automatically selecting the valuable datasets to store, our strategy can significantly reduce the total cost of the pulsar searching workflow.

VI. RELATED WORKS

Comparing to the distributed computing systems like cluster and grid, a cloud computing system has a cost benefit [4]. Assunção et al. [5] demonstrate that cloud computing can extend the capacity of clusters with a cost benefit. Using Amazon clouds' cost model and BOINC volunteer computing middleware, the work in [16] analyses the cost benefit of cloud computing versus grid computing. The idea of doing science on the cloud is not new. Scientific applications have already been introduced to cloud computing systems. The Cumulus project [22] introduces a scientific cloud architecture for a data centre, and the Nimbus [15] toolkit can directly turns a cluster into a cloud which has already been used to build a cloud for scientific applications. In terms of the cost benefit, the work by Deelman et al. [11] also applies Amazon clouds' cost model and demonstrates that cloud computing offers a cost-effective way to deploy scientific applications. The above works mainly focus on the comparison of cloud computing systems and the traditional distributed computing paradigms, which shows that applications running on cloud have cost benefits. However, our work studies how to reduce the cost if we run scientific workflows on the cloud. In [11], Deelman et al. present that storing some popular intermediate data can save the cost in comparison to always regenerating them from the input data. In [2], Adams et al. propose a model to represent the trade-off of computation cost and storage cost, but have not given the strategy to find this trade-off. In our paper, an innovative intermediate data storage strategy is developed to reduce the total cost of scientific cloud workflow systems by finding the trade-off of computation cost and storage cost. This strategy can automatically select the most appropriate intermediate data to store, not only based on the generation cost and usage rate, but also the dependency of the workflow intermediate data.

The study of data provenance is important in our work. Due to the importance of data provenance in scientific applications, much research about recording data provenance of the system has been done [13] [6]. Some of them are especially for scientific workflow systems [6]. Some popular scientific workflow systems, such as Kepler [17], have their own system to record provenance during the workflow execution [3]. In [20], Osterweil et al. present how to generate a Data Derivation Graph (DDG) for the execution of a scientific workflow, where one DDG records the data provenance of one execution. Similar to the DDG, our IDG is also based on the scientific workflow data provenance, but it depicts the dependency relationships of all the intermediate data in the system. With the IDG, we know where the intermediate data are derived from and how to regenerate them.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, based on an astrophysics pulsar searching workflow, we have examined the unique features of intermediate data management in scientific

cloud workflow systems and developed a novel cost-effective strategy that can automatically and dynamically select the appropriate intermediate datasets of a scientific workflow to store or delete in the cloud. The strategy can guarantee the stored intermediate datasets in the system are all necessary, and can dynamically check whether the regenerated datasets need to be stored, and if so, adjust the storage strategy accordingly. Simulation results of utilising this strategy in the pulsar searching workflow indicate that our strategy can significantly reduce the total cost of the scientific cloud workflow system.

Our current work is based on Amazon's cloud cost model and assumed all the application data are stored in its cloud service. However, sometimes scientific workflows have to run distributed, since some application data are distributed and may have fixed locations. In these cases, data transfer is inevitable. In the future, we will develop some data placement strategies in order to reduce data transfer among data centres. Furthermore, to wider utilise our strategy, model of forecasting intermediate data usage rate need to be studied. It must be flexible that can adapt in different scientific applications.

ACKNOWLEDGEMENT

The research work reported in this paper is partly supported by Australian Research Council under Linkage Project LP0990393. We are also grateful for the discussions with Dr. W. van Straten and Ms. L. Levin from Swinburne Centre for Astrophysics and Supercomputing on the pulsar searching process.

REFERENCE

- [1] "Amazon Elastic Computing Cloud, <http://aws.amazon.com/ec2/>", accessed on 28 Jan. 2010.
- [2] I. Adams, D. D. E. Long, E. L. Miller, S. Pasupathy, and M. W. Storer, "Maximizing Efficiency By Trading Storage for Computation," in *Workshop on Hot Topics in Cloud Computing (HotCloud'09)*, pp. 1-5, 2009.
- [3] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance Collection Support in the Kepler Scientific Workflow System," in *International Provenance and Annotation Workshop*, pp. 118-132, 2006.
- [4] 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 28 Jan. 2010.
- [5] M. D. d. Assuncao, A. d. Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *18th ACM International Symposium on High Performance Distributed Computing*, Garching, Germany, pp. 1-10, 2009.
- [6] Z. Bao, S. Cohen-Boulakia, S. B. Davidson, A. Eyal, and S. Khanna, "Differencing Provenance in Scientific Workflows," in *25th IEEE International Conference on Data Engineering, ICDE '09.*, pp. 808-819, 2009.
- [7] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: a survey," *ACM Comput. Surv.*, vol. 37, pp. 1-28, 2005.
- [8] 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.

- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] P. Groth and L. Moreau, "Recording Process Documentation for Provenance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 1246-1259, 2009.
- [14] 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.
- [15] 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.
- [16] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of Cloud Computing versus desktop grids," in *IEEE International Symposium on Parallel & Distributed Processing, IPDPS'09*, pp. 1-12, 2009.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] L. J. Osterweil, L. A. Clarke, A. M. Ellison, R. Podorozhny, A. Wise, E. Boose, and J. Hadley, "Experience in Using A Process Language to Define Scientific Workflow and Generate Dataset Provenance," in *16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Atlanta, Georgia, pp. 319-329, 2008.
- [21] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Rec.*, vol. 34, pp. 31-36, 2005.
- [22] 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.
- [23] A. Weiss, "Computing in the Cloud," *ACM Networker*, vol. 11, pp. 18-25, 2007.