Minimum Cost Benchmarking 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 in scientific cloud workflows can be stored by different storage strategies 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. With the IDG, deleted intermediate data can be regenerated, and as such we develop a novel algorithm that can develop a minimum cost storage strategy of the intermediate data in scientific cloud workflows systems. The strategy achieves the best trade-off of computation cost and storage cost by automatically storing the most appropriate intermediate datasets in the cloud storage. This strategy can be deemed as a minimum cost benchmark for all other intermediate data storage strategies in the cloud. We utilise Amazon’s cost model and apply the algorithm to general random as well as specific astrophysics pulsar searching scientific workflows for evaluation. The results show that benchmarking effectively demonstrates the cost effectiveness of other representative storage strategies.

1. Introduction

Scientific applications are usually complex and data-intensive. In many fields, like astronomy [11], 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 [9]. 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 [9].

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. [10] 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 [2] [27]. Some strategies have already been proposed to cost-effectively store the intermediate data in scientific cloud workflow systems [27].

In this paper, we propose an algorithm that can calculate the minimum cost for the intermediate data storage of scientific cloud workflow systems. 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. Then we can decide whether to store or delete an intermediate dataset to reduce the system cost. Based on the IDG, we propose an algorithm that can find the minimum cost storage strategy of the intermediate datasets of the scientific cloud workflow systems. This minimum cost can be utilised as a benchmark to evaluate the cost effectiveness of other intermediate data storage strategies.

The reminder of this paper is organised as follows. Section 2 gives a motivating example of scientific workflow and analyses the research problems. Section 3 introduces some important related concepts to our algorithm. Section 4 presents the detailed minimum cost algorithms. Section 5 demonstrates the simulation results and the evaluation. Section 6 discusses the related work. Section 7 is a discussion about the data transfer cost among cloud service providers. Section 8 addresses our conclusions and future work.

2. 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.

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-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

¹ ATNF refers to the Australian Telescope National Facility.
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. Storing different intermediate data will lead a different system cost. Traditionally, intermediate data storage strategies may be developed based on different factors, such as data’s security, users’ preference and etc., but in scientific cloud workflow systems, all the strategies should deem system cost as an important factor. Hence a benchmark of the minimum system cost is needed to evaluate the cost effectiveness of different intermediate data storage strategies. 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 and a dataset may be used by many users. Hence the datasets’ usage rates should be discovered and obtained from the system log, not only manually set by the users. Accordingly, the minimum cost intermediate data storage benchmark should be calculated based on the datasets’ usage rates in the whole system.

Hence, we need an algorithm to find the minimum cost strategy for the intermediate data storage in the scientific cloud workflow systems. This strategy can be used as a benchmark to evaluate the cost effectiveness of all other intermediate data storage strategies.

3. Cost oriented intermediate data storage in scientific cloud workflows

3.1 Data management in scientific cloud workflows

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 traditional scientific workflow systems in data 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, 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. 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 reanalysis 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 scientific applications they also can be the raw data collected from the devices. These data are the original data for processing or analysis which are usually the input of the applications. The most important
feature of these data is that if they are 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 which 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 their 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 an algorithm that can find the minimum cost strategy for intermediate data storage in the scientific cloud workflow systems.

### 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. On the contrary, with data provenance information, the regeneration of the demanding dataset may start from some stored intermediated datasets. In the scientific cloud workflow system, data provenance is recorded along the workflow execution. Taking the 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)](image)

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 $d_2$; and $d_2$ is pointed to $d_3$ and $d_5$ means $d_2$ is used to generate either $d_3$ or $d_5$ based on different operations; dataset $d_4$ and $d_6$ are pointed to $d_7$ means $d_4$ and $d_6$ are used together to generate $d_7$. $d_5$ and $d_6$ are two special datasets. $d_s$ is the start node of the IDG, which denotes all the input datasets in the system. $d_e$ is the end node of the IDG, which is pointed by all the final result datasets. 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 predecessors of the demanding data, so they can be regenerated from their nearest existing predecessor datasets.
To better describe the relationships of datasets in the IDG, we define two symbols, $\mapsto$ and $\perp$: $\mapsto$, denotes that two datasets have a generation relationship, where $d_i \mapsto d_j$ means $d_i$ is a predecessor dataset of $d_j$ in the IDG. For example, in Figure 3’s IDG, we have $d_1 \mapsto d_2$, $d_1 \mapsto d_4$, $d_3 \mapsto d_5$, $d_1 \mapsto d_7$, etc. Furthermore, $\mapsto$ is transitive, where $d_i \mapsto d_j \mapsto d_k \iff d_i \mapsto d_j \land d_j \mapsto d_k \Rightarrow d_i \mapsto d_k$. Especially, $\forall d_i \in IDG \Rightarrow d_s \mapsto d_i \mapsto d_e$.

$\perp$, denotes that two datasets do not have a generation relationship, where $d_i \perp d_j$ means $d_i$ and $d_j$ are in different branches in the IDG. For example, in Figure 3’s IDG, we have $d_3 \perp d_5$, $d_3 \perp d_6$, etc. Furthermore, $\perp$ is commutative, where $d_i \perp d_j \Leftrightarrow d_j \perp d_i$. Especially, $\exists d_i \in IDG \Rightarrow d_s \perp d_i \perp d_e$.

### 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]. For example, Amazon cloud services’ prices are as follow:

- $0.15$ per Gigabyte per month for the storage resources.
- $0.1$ per CPU hour for the computation resources.

Furthermore, the cost of data transfer is also considered, such as in Amazon’s cost model. In [10], 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:

$$\text{Cost} = C+S,$$

where the total cost of the system, $\text{Cost}$, 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.

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: $<x_i, y_i, f_i, t_i, provSet_i, CostR_i>$, where

- $x_i$, denotes the generation cost of dataset $d_i$ from its direct predecessors. To calculate this generation cost, we have to multiply the time of generating dataset $d_i$ by the price of computation resources. Normally the generating time can be obtained from the system log. For the start dataset $d_s$ and the end dataset $d_e$, $x_s=x_e=0$.

- $y_i$, denotes the cost of storing dataset $d_i$ in the system per time unit. This storage cost can be calculated by multiplying the size of dataset $d_i$ and the price of storage resources. For the start dataset $d_s$ and the end dataset $d_e$, $y_s=y_e=0$.

- $f_i$, is a flag, which denotes the status whether this dataset is stored or deleted in the system.
$t_i$, 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.

$provSet_i$, denotes the set of stored provenance datasets that are needed when regenerating dataset $d_i$, in another word, it is the set of stored predecessor datasets that are adjacent to $d_i$ in the IDG. If we want to regenerate $d_i$, we have to find its direct predecessors, which may also be deleted, so we have to further find the stored predecessors of datasets $d_i$. $provSet_i$ is the set of the nearest stored predecessors of $d_i$ in the IDG. Figure 4 shows the $provSet$ of a dataset in different situations.

**Figure 4. The datasets’ provSet in a general IDG**

Formally, we can describe a dataset $d_i$’s $ProvSet$ as follow:

$provSet_i = \{d_j | \forall d_j \in IDG \land f_j = "stored" \land d_j \leftrightarrow d_i \land \neg \exists d_k \in IDG \land d_k \leftrightarrow d_i \lor \exists d_k \in IDG \land d_k \leftrightarrow d_i \land f_k = "deleted"\}$

$provSet$ is a very important attribute of a dataset in calculating its generation cost. When we want to regenerate a dataset in the IDG, we have to start the computation from the dataset in its $provSet$. Hence, for dataset $d_i$, its generation cost is:

$genCost(d_i) = x_i + \sum_{d_j \in provSet_i \land d_j \leftrightarrow d_i} x_k$

This cost is a total cost of 1) the generation cost of dataset $d_i$ from its direct predecessor datasets and 2) the generation costs of $d_i$’s deleted predecessors that need to be regenerated.

$CostR_i$, is $d_i$’s cost rate, which means the average cost per time unit of the dataset $d_i$ in the system. If $d_i$ is a stored dataset, then $CostR_i = y_i$. 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. $CostR_i = genCost(d_i)/t_i$. The storage statuses of the datasets have strong influence on their cost rate. If $d_i$’s storage status is changed, not only the cost rate of itself, $CostR_i$, will change, but also the generation cost of $d_i$’s successors will change correspondingly.

Hence, the system cost rate of managing intermediate data is the sum of $CostR$ of all the intermediate datasets, which is $\sum_{d_i \in IDG} CostR_i$. Given a 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:

$\text{Total Cost} = \int_{T_0}^{T_n} \left( \sum_{d_i \in IDG} CostR_i \right) \, dt$

Based on the definition of the dataset’s cost rate, the system’s cost rate is highly depends on the storage strategy of the intermediate datasets. Storing different intermediate datasets will lead
different cost rate to the system. In scientific cloud workflow systems, intermediate data storage strategies should try to reduce this cost rate.

4. Minimum cost benchmarking of intermediate data storage

Based on the cost model discussed in section 3, in scientific cloud workflow systems, the system cost rate may differ a lot with different intermediate data storage strategies. In this section, we will introduce an algorithm that can find the minimum cost strategy of storing intermediate datasets in the system, based on a given IDG. This minimum cost can be used as a benchmark to evaluate the cost of different intermediate data storage strategies.

The basic idea of finding the minimum cost strategy is that: 1) we add edges between every two datasets and construct a Cost Graph (CG). In the CG, the paths from the start dataset to the end dataset have a one-to-one mapping to the storage strategies, and the length of the path equals the system cost rate. 2) We use the well known Dijkstra algorithm to find the shortest path, which is the minimum cost storage strategy. To describe the algorithm, we start with the calculation of minimum cost for the single branch IDG, and then expand it to the general complex IDGs.

4.1 Minimum cost algorithm for single branch IDG

Single branch IDG means the IDG is a straight line, where all the datasets in the IDG only have one predecessor and one successor except the first and last datasets.

Given a single branch IDG, which has datasets \( d_1, d_2 \ldots d_n \). We construct the corresponding CG by the followed three steps:

1) We add new edges to the IDG. For every dataset in the IDG, we add edges that start from it and point to all its successors. Formally, for a dataset \( d_i \) it has out-edges to all the datasets in the set of \( \{ d_j \mid d_j \in IDG \land d_i \rightarrow d_j \} \), and it has in-edges from all the datasets in the set of \( \{ d_k \mid d_k \in IDG \land d_k \rightarrow d_i \} \). Hence, for any two datasets in the IDG, we have an edge between them. If we denote the edge from dataset \( d_i \) to \( d_j \) as \( e_{d_i, d_j} \), then we have \( \forall d_i, d_j \in IDG \land d_i \rightarrow d_j \Rightarrow \exists e_{d_i, d_j} \).

2) We set weights to the edges. The reason we call the graph Cost Graph is because the weights of its edges are composed by the cost rates of the datasets. For an edge \( e_{d_i, d_j} \), we denote its weight as \( \omega_{d_i, d_j} \), which is defined as the sum of cost rates of \( d_j \) and the datasets between \( d_i \) and \( d_j \), supposing that only \( d_i \) and \( d_j \) are stored and rest of the datasets between \( d_i \) and \( d_j \) are all deleted. Formally, \( \omega_{d_i, d_j} = y_j + \sum_{d_k \in IDG \land d_k \rightarrow d_j \rightarrow d_{k'}} (\text{genCost}(d_k)/t_k) \). Since we are discussing the single branch IDG, for the datasets between \( d_i \) and \( d_j \), \( d_i \) is the only dataset in their provSet. Hence we can further get: \( \omega_{d_i, d_j} = y_j + \sum_{d_k \in IDG \land d_i \rightarrow d_{k'}} (\text{genCost}(d_k)/t_k) \).

In Figure 5, we demonstrate a simple example of constructing the CG for an IDG that only have three datasets.
From the construction steps, we can clearly see that the CG is a acyclic complete oriented graph, also called transitive tournament in graph theory, in which \( d_s \) is the start dataset that only has out-edges and \( d_e \) is the end dataset that only has in-edges. We use the Dijkstra algorithm to find the shortest path from \( d_s \) to \( d_e \). Dijkstra algorithm is a classic greedy algorithm to find the shortest path in graph theory. It starts from \( d_s \), and gradually finds the shortest path to all the datasets in the CG, until it reaches \( d_e \). We denote the shortest path from \( d_s \) to \( d_e \) as \( P_{\text{min}} \).

**Theorem:** Given a single branch IDG, the length of shortest path \( P_{\text{min}} \) of its CG is the minimum cost rate of the system to store the Intermediate datasets, and the corresponding storage strategy is to store the datasets that \( P_{\text{min}} \) has passed.

**Proof:**

1) There is a one-to-one mapping between storage strategies and the paths from \( d_s \) to \( d_e \) in the CG. Given any storage strategy of an IDG that has \( n \) dataset, we suppose \( m \) datasets are stored, where \( 0 \leq m \leq n \). Because the IDG is a single branch one, we can find an order of these stored datasets, where \( d_s \mapsto d_s' \mapsto d_s'' \mapsto \ldots \mapsto d_m' \mapsto d_e \), and because \( \forall d_i, d_j \in IDG \land d_i \mapsto d_j \Rightarrow \exists e < d_i, d_j > \), we can find the exact path in the CG that has passed all the stored datasets. Similarly, given any path from \( d_s \) to \( d_e \) in the CG, we can find the datasets it has passed, and because the CG is an acyclic graph, this is the only path that can pass these datasets. We can get the exact storage strategy of storing these datasets. Hence the paths and the strategies have a one-to-one mapping.

2) The length of a path from \( d_s \) to \( d_e \) in the CG equals the system cost rate of the corresponding storage strategy. Given any path \( P \) from \( d_s \) to \( d_e \) in the CG, we suppose it passes the datasets \( d_s \mapsto d_s' \mapsto d_s'' \mapsto \ldots \mapsto d_m' \mapsto d_e \). Hence the length of \( P \) is the sum of all the edges:

\[
P < d_s, d_e > = \sum_{i=1}^{m} \left( y_i + \sum_{d_j \in IDG \land d_j \mapsto d_i} (\text{genCost}(d_j)/t_k) \right) + \sum_{i=1}^{m} \left( y_i + \sum_{d_j \in IDG \land d_j \mapsto d_{i+1}} (\text{genCost}(d_j)/t_k) \right)
\]

As defined in Section 3, the system cost rate is \( \sum_{i \in IDG} \text{Cost}_i \).

3) \( P_{\text{min}} \) is the shortest path from \( d_s \) to \( d_e \).

Because of 1), 2) and 3), theorem holds.

Furthermore, in the process of Dijkstra algorithm, if a dataset \( d_i \) is found, we know the shortest path from \( d_s \) to \( d_i \). This shortest path represent the corresponding minimum cost storage strategy of the datasets \( d_i \) and \( \{ d_k | \forall d_k \in IDG \land d_k \mapsto d_i \} \), where the length of this path is the total cost rate of these datasets.

Based on the discussion above, the minimum cost storage strategy of the single branch IDG can be found.

4.2 Minimum cost algorithm for the IDG with one multi-branch block

Single branch IDG is a special case of the IDGs. In the real world, intermediate datasets generated in scientific workflows may have complex relationships, such that different datasets may be generated from a same datasets by different operations, and different datasets may be used together to generate one datasets. In another word, the IDG may have branches. The method we introduce in Section 4.1 can only find the minimum cost strategy for the single branch IDG, since the Dijkstra algorithm will only choose one branch as the shortest path to reach the end dataset in
the IDG that has multiple branches. For the rest branches, we do not know the storage strategies.
In this section, based on the algorithm proposed in Section 4.1, we will further design an
algorithm to find the minimum cost storage strategy for the IDG that has one block of multiple
branches.

4.2.1 Classification of the cost edges based on the block

A block in an IDG is defined as a set of datasets in the IDG that contains several parallel sub
single branch IDGs, where the first datasets of these sub IDGs are generated from a common
dataset and the last datasets of these sub IDGs generate another common dataset together. In
another word, a block is the set of parallel sub single branch IDGs that split from a common
dataset and merge in another common dataset. We denote the block as \( B \), and the sub single
branch IDGs as \( Br_1, Br_2...Br_m \), where \( B = \{ Br_1 \cup Br_2 \cup ... Br_m \} \).

![Figure 6. Different types of the cost edges in the IDG with a block](image)

To find the minimum cost storage strategy of the IDG with a block, we still use the method
introduced in section 4.1 that constructs a CG for the IDG. Figure 6 shows the IDG in Figure 3,
which has a block \( B = \{ d_3, d_4, d_5, d_6 \} \), sub single branch IDGs \( Br_1 = \{ d_3, d_4 \} \), \( Br_2 = \{ d_5, d_6 \} \).

At first, we add the cost edges to the IDG based on the same rule in section 4.1, which is
\( \forall d_i, d_j \in IDG \land d_i \mapsto d_j \Rightarrow \exists e < d_i, d_j > \). In Figure 6, we only draw some representative edges of the
CG. Due to the existence of the block, the edges can be classified into three categories. The
definition of this classification is as follow:

- **in-block edge**: \( e < d_i, d_j > \) is an in-block edge means that the edge starts from \( d_i \), which is a
dataset outside of the block, and points to \( d_j \), which is a dataset in the block, such as \( e < d_2, d_3 > \), \( e < d_2, d_5 > \), \( e < d_1, d_5 > \) in Figure 6. Formally, we define \( e < d_i, d_j > \) is an in-block edge,
where \( \exists d_k \in IDG \land d_i \mapsto d_k \land d_j \mapsto d_k \).

- **out-block edge**: \( e < d_i, d_j > \) is an out-block edge means that the edge starts from \( d_i \), which is a
dataset in the block, and points to \( d_j \), which is a dataset outside of the block, such as \( e < d_4, d_6 > \), \( e < d_6, d_7 > \), \( e < d_3, d_8 > \) in Figure 6. Formally, we define \( e < d_i, d_j > \) is an out-block edge,
where \( \exists d_k \in IDG \land d_i \mapsto d_k \land d_j \mapsto d_k \).

- **ordinary edge**: \( e < d_i, d_j > \) is an ordinary edge means that datasets \( d_i \) and \( d_j \) are both in the
block, or both out of the block, such as \( e < d_5, d_7 > \), \( e < d_3, d_4 > \), \( e < d_1, d_8 > \), etc. in Figure 6.
Formally, we define \( e < d_i, d_j > \) is an ordinary edge,
where \( \neg \exists d_k \in IDG \land \left[ ( d_i \mapsto d_k \land d_j \mapsto d_k ) \lor ( d_i \mapsto d_k \land d_j \mapsto d_k ) \right] \).

4.2.2 Setting the weights to different types of edges

Next, we set weights to the edges. As defined in Section 4.1, the weight of the edge \( e < d_i, d_j > \)
is the sum of cost rates of \( d_j \) and the datasets between \( d_i \) and \( d_j \), supposing that only \( d_i \) and \( d_j \) are
stored and rest of the datasets between \( d_i \) and \( d_j \) are all deleted. \( \omega_{ij} = y_j + \sum_{d_k \mapsto d_i \mapsto d_j} \left( genCost(d_k) / t_k \right) \) In the IDG with a block, this rule is still applicable to the
ordinary edges and the in-block edges, since given an edge \( e < d_i, d_j > \), if it is an ordinary edge or in-block edge, \( d_i \) is the provSet of all the datasets between \( d_i \) and \( d_j \), which is the same as the single branch IDG. Hence the weights of these edges can also be calculated in the same way as the single branch IDG. However, for the out-block edges, the situation is different that the rule in section 4.1 is not suitable for calculating their weights anymore. Suppose \( e < d_i, d_j > \) is an out-block edge, where \( d_i \in B \) and \( d_j \notin B \).

1) Some datasets between \( d_i \) and \( d_j \), may have more than one datasets in their provSet. These datasets are \( \{ d_i \mid \forall d_p \in B \land d_p \mapsto d_k \mapsto d_j \} \), which means the datasets that are between \( d_i \) and \( d_j \), and also outside of the block. The regeneration of these datasets not only need \( d_i \), but also need the stored provenance datasets from other branches of the block. For example, following the formula in Section 4.1, the weight of the out-block edge \( e < d_3, d_8 > \) in Figure 6 is \( \omega < d_3, d_8 >= y_8 + genCost(d_4)/t_4 + genCost(d_5)/t_5 \), where if we want to calculate \( genCost(d_7) \), we also have to know the storage status of \( d_5 \) and \( d_6 \). Hence, to calculate the weight of an out-block edge, we have to know the storage strategies of all the branches in the block.

2) If we use the same method in section 4.1 to set the weight of the out-block edge \( e < d_i, d_j > \), the path that contains \( e < d_i, d_j > \) across the block in the CG can not represent the storage strategy of all the datasets in the block. This is because the datasets in \( \{ d_i \mid \forall d_k \in B \land d_k \mapsto d_j \} \) are not represented in the path. For example, in Figure 6, the path from \( d_s \) to \( d_8 \) that contains the out-block edge \( e < d_3, d_8 > \), does not represent the storage status of datasets \( d_5 \) and \( d_6 \), and the length of the path also does not contain the cost rates of \( d_5 \) and \( d_6 \). Hence, to maintain the mapping between the paths and the storage strategies, the weight the out-block edge \( e < d_i, d_j > \) should contain the cost rates of the datasets in \( \{ d_i \mid \forall d_k \in B \land d_k \mapsto d_j \} \).

Based on the 2 points above, we define the weight of the out-block edge \( e < d_i, d_j > \) as

\[
\omega < d_i, d_j >= y_i + \sum_{\{d_k\mid \forall d_k \in IDG, d_k \mapsto d_i \}} (genCost(d_k)/t_k) + \sum_{\{d_k\mid \forall d_k \in IDG, d_k \mapsto d_j \}} CostR_k.
\]

In this formula, \( \sum_{\{d_k\mid \forall d_k \in IDG, d_k \mapsto d_i \}} CostR_k \) means the total cost rates of the datasets that are in the different branches of dataset \( d_i \) in the block. The Dijkstra algorithm we use to find the shortest path will only choose one branch in the block as the path. This formula guarantees that the length of the path with an out-block edge \( e < d_i, d_j > \) still equals the total cost rate of the datasets, which is \( P < d_i, d_j >= \sum_{\{d_k\mid \forall d_k \in IDG, d_k \mapsto d_i \}} CostR_k \). If we want to further guarantee the length of the path is the minimum total cost rate of the system, we have to set \( \sum_{\{d_k\mid \forall d_k \in IDG, d_k \mapsto d_i \}} CostR_k \) with the minimum value. Hence we need to find the minimum cost storage strategy of the datasets that are in the different branches of dataset \( d_i \) in the block.

However, because of the dependencies between datasets in the IDG, given different stored adjacent predecessor and successor of a block, the minimum cost storage strategy may be different. For an out-block edge \( e < d_i, d_j > \), the value of \( \sum_{\{d_k\mid \forall d_k \in IDG, d_k \mapsto d_i \}} CostR_k \) in its weight would be different according to different dataset which is given as the adjacent stored predecessor of the block. Hence, we have to create multiple CG for the IDG that has a block, in order to find the minimum cost storage strategy.

### 4.2.3 Steps of finding minimum cost storage strategy for an IDG with one block

For the IDG with one block, the basic idea of finding its minimum cost storage strategy is same as the single branch IDG, which is constructing CG to the IDG and using the Dijkstra algorithm to find the shortest path. However, for the IDG with one block, we may create many CGs and the Dijkstra algorithm need to find the shortest path among all the CGs. The main steps are as follow.
**Step 1**, create the initial CG to the IDG. Based on the discussion in Section 4.2.1 and 4.2.2, we add cost edges to the IDG and set the weights for the ordinary edges and in-block edges based on the same rule in the single branch IDG. For the out-block edges, we set their weights as infinity at the initial stage. The initial CG is shown in Figure 7 (a).

**Step 2**, start the Dijkstra algorithm to find the shortest path from $d_i$. The algorithm will not stop until an in-block edge is chosen and added to the current shortest path. Whenever an in-block edge is chosen by the Dijkstra algorithm, denoted as $e^{<dp, dq>}$, we will create a new CG for the IDG, denoted as CG($d_p, B$). This process is also shown in Figure 7 (a) and (b).

**Step 3**, create CG($d_p, B$). First, we copy all the information of the initial CG to the new CG($d_p, B$), including the paths that have been found by the Dijkstra algorithm. Second, we delete all the in-block edges in CG($d_p, B$) except the edges start from $d_p$. Last, we update the weight values of all the out-block edges in CG($d_p, B$).

**Step 4**, calculate the weight of an out-block edge $e^{<di, dj>}$ in CG($d_p, B$). As discussed in Section 4.2.2, the weight of $e^{<di, dj>}$ is

$$
\omega^{<di, dj>} = y_j + \sum_{i \in \mathcal{A}_{idgi \rightarrow dj}} \left( \frac{\text{genCost}(d_i)}{t_k} \right) + \sum_{i \in \mathcal{A}_{idgi \rightarrow dj}} \text{CostR}_{ik},
$$

where we have to find the minimum value of $\sum_{i \in \mathcal{A}_{idgi \rightarrow dj}} \text{CostR}_{ik}$.

From Figure 7 (b) we can see that the block $B$ is composed of parallel branches, i.e. $B = \{Br_1 \cup Br_2 \cup ...Br_n\}$. Each branch in block $B$ can be deemed as a sub single branch IDG, so that its minimum cost storage strategy can be calculated using the algorithm in section 4.1, given that $d_p$ is the start dataset and $d_j$ is the end dataset. Furthermore, this minimum cost storage strategy...
should be under the condition that the datasets in \(\{d_i \mid \forall d_k \in IDG \land (d_p \mapsto d_i \mapsto B \lor B \mapsto d_i \mapsto d_j)\}\) are all deleted, which are the datasets between \(d_p\) and \(B\), between \(B\) and \(d_i\). Hence we delete all the edges that end at these datasets. The final CG of a sub single branch IDG of block \(B\) is also demonstrated in Figure 7 (c).

\[
\sum_{[d_k \mid \forall d_k \in IDG \land d_k \not \in \{d_i\}] \bigwedge \exists \text{Cost}_{R_h}\]

is the sum of the cost rates of all the branches except the branch that contains \(d_i\), hence its minimum value is the sum of minimum cost rates of these branches, where the branches’ minimum cost storage strategies can be calculated. Hence the weight of an out-block edge can be calculated.

Step 5, use Dijkstra algorithm to find the shortest path in all created CGs. Whenever an in-block edge is chosen in the initial CG, go to step 3. If the ending dataset \(d_e\) is reached in one CG, the algorithm stops. The shortest path which is found in that CG represents the minimum cost storage strategy of the IDG.

4.3 Minimum cost algorithm for general IDG

In the real world applications, the IDG’s structure could be complex, i.e. there may exist more than one blocks in an IDG. However, to calculate the minimum cost storage strategy of an IDG, no matter how complex the IDG’s structure is, we can reduce the process to the calculations of single branch IDGs and the IDGs with one block. In this section we will discuss how to calculate the minimum cost storage strategy for a general IDG, where there are two new situations that we have to deal with.

First is the IDG that has two serial blocks. When constructing the CGs for this kind of IDGs, there is a new type of cost edge appeared, i.e. the edge across two blocks. For example, in Figure 8, \(e \prec d_i, d_j\) is this kind of edge, where it is an out-block edge of block \(B_1\), and also an in-block edge of block \(B_2\).

![Figure 8. CG for a general IDG with two serial blocks](image)

In the searching of shortest path in the CGs of this kind of IDG that has two serial blocks, if the edge \(e \prec d_i, d_j\) is found by the Dijkstra algorithm, we have to create a new CG\((d_i, B_2)\), since \(e \prec d_i, d_j\) is an in-block edge of \(B_2\). However, because \(e \prec d_i, d_j\) is also an out-block edge, \(d_i\) is not the only dataset in \(d_i\)'s provSet\(_j\), provSet\(_j\) may contain different datasets from other branches of block \(B_1\), which will be used for calculation the weight of the out-block edges of block \(B_2\). Hence, we denoted the created cost graph of \(e \prec d_i, d_j\) as CG\((\text{provSet}_j, B_2)\). If the Dijkstra algorithm find another edge from \(d_i\) to block \(B_2\) after \(e \prec d_i, d_j\) is found, which is \(e \prec d_i, d_k\) shown in Figure 8. We will compare \(d_i\)'s provSet\(_j\) with provSet\(_j\). If provSet\(_j\) dose not equal provSet\(_k\), we will create new CG\((\text{provSet}_k, B_2)\). Rest of the edges will be treated in the same way as the IDG with one block.

Second is the IDG that has two nested blocks. When constructing the CGs for this kind of IDGs, there is also a new type of cost edge appeared, i.e. the edge is an in-block edge of two nested blocks. For example, in Figure 9, \(e \prec d_i, d_j\) is this kind of edge, where it is an in-block edge of both blocks \(B_1\) and \(B_2\).
5. Evaluation

The minimum cost benchmarking for intermediate data storage strategies proposed in this paper is generic. It can be used in any scientific workflow applications. In this section, we demonstrate the simulation results that we conduct on Swinburne high performance supercomputing facility. In the beginning, we use random workflows and datasets to demonstrate the comparison of our benchmark with different storage strategies. Then we deploy our algorithm to the pulsar searching workflow described in Section 2, and use the real world statistics to demonstrate how our algorithm finds the minimum cost strategy in storing the intermediate datasets of the pulsar searching workflow.

5.1 Simulation environment and strategies

Figure 10 shows the structure of our simulation environment. 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. SwinDeW-C [26] (Swinburne Decentralised Workflow for Cloud) is a cloud workflow system developed based on SwinDeW [24] and SwinDeW-G [25]. It is currently running on our simulation cloud that can interpret and execute workflows, send and retrieve, save and delete data in the virtual data centres. Through a user interface at the application level, which is a web portal, we can deploy workflows and upload application data to the cloud. In the simulation, we facilitated our algorithm in SwinDeW-C to find the minimum cost benchmark of storing intermediate datasets in the simulation cloud.
To evaluate the effectiveness of our minimum cost benchmark, we compare the total system costs of different storage strategies with our benchmark. The strategies are: 1) store the intermediate datasets based on their usage rates; 2) store the intermediate datasets based on their generation costs; 3) dependency based cost-effective strategy reported in [27].

We have run a large number of simulations with different parameters to evaluate the performance of our benchmark. We will evaluate some representative results in this section. More simulation results, as well as the program code, are available at http://www.swinflow.org/docs/Benchmark.zip, where readers can download and view the results and implementation themselves.

5.2 Random simulations and results

To evaluate the overall performance of our strategy, we ran a large number of random simulations with the three strategies introduced above. In the random simulations, we use random generated workflows to construct the IDG, and give every intermediate dataset random size, generation time, usage rate, and then run the workflows under different pricing models. We compare the total system cost over 30 days of different strategies, which shows the cost effectiveness of the strategies comparing to our minimum cost benchmark.

We pick one test case as representative. In this case, we let the workflow randomly generate 50 intermediate datasets, each with a random size from 100GB to 1TB. The generation time is also random, from 1 hour to 10 hours. The usage rate (time between every usage) is again randomly from 1 day to 10 days. 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. We run our algorithm on this IDG to calculate the minimum cost strategy, where 9 of the 50 datasets are chosen to be stored. We use this minimum cost strategy as benchmark to evaluate the strategies introduced in Section 5.1. More random simulation cases can also be found in the link given in Section 5.1.
Figure 11 shows the comparison of the minimum cost benchmark with the strategy of storing high generation cost datasets. We compare the total system costs over 30 days of the strategies that store different percentage of datasets based on the generation cost, and the minimum cost benchmark. The two extreme strategies of storing all the datasets and deleting all the datasets are also included. In Figure 11, we can clearly see the cost effectiveness of different strategies comparing with the benchmark, where storing top 10% generation cost datasets turns to be the most cost-effective strategy in this case. But the system cost is still very much higher than the minimum cost benchmark.

![Figure 11. Cost effectiveness evaluation of the “store high generation cost datasets” strategy](image)

Then we will evaluate the storing often used datasets strategy by comparing with the benchmark. We still run simulation of strategies that storing different percentage of datasets based on their usage rates. Figure 12 shows the comparison of the total system costs over 30 days, where we can clearly see the cost effectiveness of different strategies comparing with the benchmark. Also, the strategy of storing top 10% generation cost datasets turns to be the most cost-effective one in this case. Comparing to figure 11, the strategy of storing often used datasets is more cost-effective than storing high generation cost dataset, but it is still much higher than the minimum cost benchmark.

![Figure 12. Cost effectiveness evaluation of the “store often used datasets” strategy](image)
The intermediate data storage strategies reported in [27] are also based on the IDG, which have considered the data dependencies in calculating the datasets’ generation cost and storage cost. Figure 13 show the comparison of the dependency based strategies with the minimum cost benchmark. In the static strategy, datasets’ storage status are decided when they are first generated in the system by comparing their generation cost rate and storage cost rate, and in the dynamic strategy, whenever the datasets are regenerated in the system, their storage status will be recalculated and dynamically changed, and other datasets’ storage status may also be adjusted accordingly. In Figure 13, we can see that the system cost of the dependency based strategies, especially the dynamic one, are close to the minimum cost benchmark, which are more cost-effective than the strategies in Figures 11 and 12.

Figure 13. Cost effectiveness evaluation of the dependency based strategy

5.3 Pulsar case simulation and results

The random simulations demonstrate how to use our minimum cost benchmark to evaluate the cost effectiveness of different data storage strategies. Next we utilised it for the pulsar searching workflow we introduced in Section 2 and show how the benchmark works in the real scientific application.

In the pulsar example, during the workflow execution, six intermediate datasets are generated. The IDG of this pulsar searching workflow is shown in Figure 14, 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.

Figure 14. IDG of pulsar searching workflow

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 2 days, and rest of the intermediate datasets to be used once every 5 days. Based on this setting, we run the above mentioned 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 30 days which is shown in Figure 15.

From Figure 15 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; 3-4) the costs of the “store high generation cost datasets” and “store often used datasets” strategies are in the middle range, which are much lower than the “store all” and “store none” strategies. The cost lines are only a little fluctuated, because the intermediate datasets are partially stored; 5) the benchmark is the minimum cost strategy.

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

<table>
<thead>
<tr>
<th>Strategies</th>
<th>Extracted beam</th>
<th>De-dispersion files</th>
<th>Accelerated de-dispersion files</th>
<th>Seek results</th>
<th>Pulsar candidates</th>
<th>XML files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store all</td>
<td>Stored</td>
<td>Stored</td>
<td>Stored</td>
<td>Stored</td>
<td>Stored</td>
<td>Stored</td>
</tr>
<tr>
<td>Store none</td>
<td>Deleted</td>
<td>Deleted</td>
<td>Deleted</td>
<td>Deleted</td>
<td>Deleted</td>
<td>Deleted</td>
</tr>
<tr>
<td>Store high generation cost datasets</td>
<td>Deleted</td>
<td>Stored</td>
<td>Stored</td>
<td>Deleted</td>
<td>Deleted</td>
<td>Stored</td>
</tr>
<tr>
<td>Store often used datasets</td>
<td>Deleted</td>
<td>Stored</td>
<td>Deleted</td>
<td>Deleted</td>
<td>Deleted</td>
<td>Deleted</td>
</tr>
<tr>
<td>Minimum cost benchmark</td>
<td>Deleted</td>
<td>Stored</td>
<td>Deleted</td>
<td>Stored</td>
<td>Deleted</td>
<td>Stored</td>
</tr>
</tbody>
</table>

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 benchmark represents the most appropriate strategy for the intermediate data storage which has the minimum system cost.

6. 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. [10] 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 [10], 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 [27], Yuan et al. proposed a cost-effective strategy for intermediate data storage in scientific cloud workflows systems that takes data dependency into consideration, but this strategy did not achieve the minimum cost of the system. In this paper, an innovative algorithm is proposed to calculate the minimum cost strategy of storing intermediate datasets in scientific cloud workflow systems that achieves the best trade-off of computation cost and storage cost.

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.

7. Discussion

As cloud computing is such a fast growing market, different cloud service providers will appear. In the future, we will be able to flexibly select service providers to conduct our applications based on their pricing models. An intuitive idea is to incorporate different cloud service providers in our applications, where we can store the data with the provider who has a lower price in storage resources, and choose the provider who has lower price of computation resources to run the computation tasks. However, at present, it is not practical to run scientific workflow applications among different cloud service providers, because of the following reasons:
1) The application data in scientific workflows are usually very large in size. They are too large to be transferred efficiently via the Internet. Due to bandwidth limitations of the Internet, in today’s scientific projects, delivery of hard disks is a very common way to transfer application data, and it is also considered to be the most efficient way to transfer terabytes of data [4]. Nowadays, express delivery companies can deliver the hard disks nation wide by the end of the next day and world wide in 2 or 3 days, by contrast, transferring one terabyte data via Internet will take more than 10 days at a speed of 1MB/s. To break the bandwidth limitation, some institutions setup dedicated fibres to transfer data. For example, Swinburne University of Technology has built a fibre to Parkes with gigabit bandwidth. However, it is mainly used for transferring gigabytes of data. To transfer terabytes of data, scientists still prefer to ship hard disks. Furthermore, building fibre connections is still expensive, and they are not wildly used in the Internet. Hence, transferring scientific application data between different cloud service providers via Internet is not efficient.

2) Cloud service providers place high cost on data transfer in and out their data centres, in contrast, data transfer within one cloud service provider’s data centres are usually free. For example, the data transfer price of Amazon cloud service is: $0.1 per GB of data transferred in and $0.17 per GB of data transferred out. Compared to the storage price of $0.15 per GB per month, the data transfer price is relatively high, such that finding a cheaper storage cloud service provider and transferring data out may not be cost-effective. In cloud service providers’ position, they charge high price on data transfer not only because of the bandwidth limitation, but also as a business strategy. As data are deemed as an important resource today, cloud service providers want users to keep all the application data in their storage cloud. For example, Amazon has made a promotion that places a zero price on data transferred into its data centres, until June 30. 2010. Which means users can upload their data to Amazon’s cloud storage for free. However, the price of data transfer out of Amazon is still the same.

Given the two points discussed above, the most efficient and cost-effective way to run scientific application in cloud is to keep all the application data and run the workflows with one cloud service provider, where the same conclusion is also stated in [10]. Hence, in the strategy stated in this paper, we did not take data transfer cost into consideration.

8. 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 algorithm that can calculate the minimum cost storage strategy for the intermediate datasets of a scientific workflow in the cloud. This strategy achieves the best trade-off of computation cost and storage cost of the cloud resources, which can be used as the minimum cost benchmark for evaluating the cost effectiveness of all other more efficient data storage strategies. Simulation results of utilising this strategy in both random workflows and the pulsar searching workflow indicate that our benchmark serves for such a purpose in scientific cloud workflow system.

Our current work is based on Amazon’s cloud cost model and assumed that all the application data are stored in its cloud service. However, sometimes scientific workflows have to run in a distributed manner, 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 widely utilise our
strategy, models of forecasting intermediate data usage rate need to be studied. It must be flexible that can be adapted to 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, as well as the simulation work and English proof reading assistance from Mr. B. Gibson.

Reference


