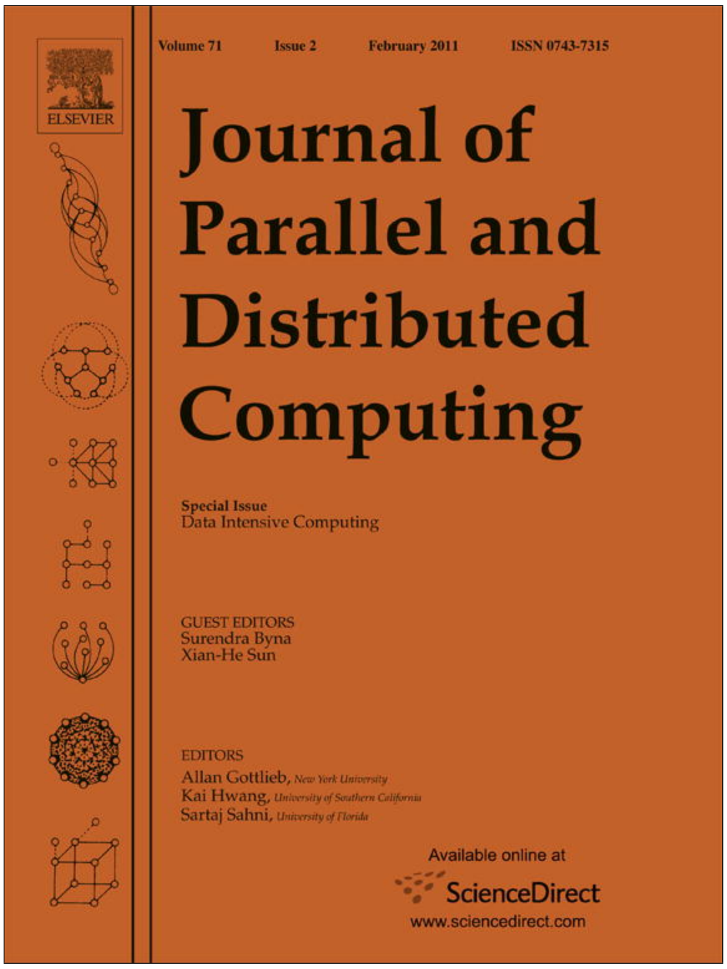

Volume 71
Issue 2
February 2011
ISSN 0743-7315
ELSEVIER
Journal of Parallel and Distributed Computing
Special Issue
Data Intensive Computing
GUEST EDITORS
Surendra Byna
Xian-He Sun
EDITORS
Allan Gottlieb, New York University
Kai Hwang, University of Southern California
Sartaj Sahni, University of Florida
Available online at
ScienceDirect
www.sciencedirect.com

J. Parallel Distrib. Comput. 71 (2011) 316–332

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

On-demand minimum cost benchmarking for intermediate dataset 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 3122, Victoria, Australia

ARTICLE INFO

Article history:
Received 23 January 2010
Received in revised form
13 August 2010
Accepted 2 September 2010
Available online 16 September 2010

Keywords:
Dataset storage
Scientific workflow
Cloud computing
Cost benchmarking

ABSTRACT

Many scientific workflows are data intensive: large volumes of intermediate datasets are generated during their execution. Some valuable intermediate datasets 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 clouds has become popular nowadays, more intermediate datasets in scientific cloud workflows can be stored by different storage strategies based on a pay-as-you-go model. In this paper, we build an intermediate data dependency graph (IDG) from the data provenances in scientific workflows. With the IDG, deleted intermediate datasets can be regenerated, and as such we develop a novel algorithm that can find a minimum cost storage strategy for the intermediate datasets in scientific cloud workflow 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 utilised on demand as a minimum cost benchmark for all other intermediate dataset storage strategies in the cloud. We utilise Amazon clouds' 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 over other representative storage strategies.

1. Introduction

Scientific applications are usually complex and data intensive. In many fields, such as astronomy [14], high-energy physics [24] and bioinformatics [27], 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 datasets will be generated [15]. They could be even larger than the original dataset(s) and contain some important intermediate results. After the execution of a scientific workflow, some intermediate datasets 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 datasets; (2) for collaboration, the intermediate results may need to be shared among scientists from different institutions and the intermediate datasets may need to be reused. Storing valuable intermediate datasets can save their regeneration cost when they are reused, not to mention the waiting time saved by avoiding regeneration. Given the large sizes of the datasets, running scientific workflow applications usually need not only high-performance computing resources but also massive storage [15].

Nowadays, popular scientific workflows are often deployed in grid systems [24] because they have high performance and massive storage. However, building a grid system is extremely expensive and it is normally not an option 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 datasets.

In late 2007, the concept of cloud computing was proposed [32] and it is deemed the next generation of IT platforms that can deliver computing as a kind of utility [11]. Foster et al. made a comprehensive comparison of grid computing and cloud computing [17]. Cloud computing systems provide high performance and massive storage required for scientific applications in the same way as grid systems, but with a lower infrastructure construction cost among many other features, because cloud computing systems are composed of data centres which can be clusters of commodity hardware [32]. Research into doing science and data-intensive applications on the cloud has already commenced [25], such as early experiences like the Nimbus [21] and Cumulus [31] projects. The work by Deelman et al. [16] shows that cloud computing offers a cost-effective solution for data-intensive applications, such as scientific workflows [20]. Furthermore, cloud computing systems offer a new model: namely, that scientists from all over the world can

[*] Corresponding author.
E-mail addresses: dyuan@swin.edu.au (D. Yuan), yyang@swin.edu.au (Y. Yang), xliu@swin.edu.au (X. Liu), jchen@swin.edu.au (J. Chen).

0743-7315/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2010.09.003

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 use of all the resources need to be paid for. For a scientific cloud workflow system, storing all the intermediated datasets generated during workflow executions may cause a high storage cost. In contrast, if we delete all the intermediate datasets and regenerate them every time they are needed, the computation cost of the system may well be very high too. The intermediate dataset storage strategy is to reduce the total cost of the whole system. The best way is to find a balance that selectively stores some popular datasets and regenerates the rest of them when needed [1,36,38]. Some strategies have already been proposed to cost-effectively store the intermediate data in scientific cloud workflow systems [36,38].

In this paper, we propose a novel algorithm that can calculate the minimum cost for intermediate dataset storage in scientific cloud workflow systems. The intermediate datasets in scientific cloud workflows often have dependencies. During 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, we know 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 logs. Then we can decide whether an intermediate dataset should be stored or deleted in order to reduce the system cost. However, the cloud computing environment is very dynamic, and the usages of intermediate datasets may change from time to time. Given the historic usages of the datasets in an IDG, we propose a cost transitive tournament shortest path (CTT-SP) based algorithm that can find the minimum cost storage strategy of the intermediate datasets on demand in scientific cloud workflow systems. This minimum cost can be utilised as a benchmark to evaluate the cost effectiveness of other intermediate dataset storage strategies.

The remainder of this paper is organised as follows. Section 2 gives a motivating example of a scientific workflow and analyses the research problems. Section 3 introduces some important related concepts and the cost model of intermediate dataset storage in the cloud. Section 4 presents the detailed minimum cost algorithms. Section 5 demonstrates the simulation results and the evaluation. Section 6 discusses 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, the Swinburne Astrophysics group has been conducting a pulsar searching survey using the observation data from the Parkes Radio Telescope, which is one of the most famous radio telescopes in the world [8]. Pulsar searching is a typical scientific application. It involves complex and time-consuming tasks and needs to process terabytes of data. Fig. 1 depicts the high-level structure of a pulsar searching workflow, which is currently running on the Swinburne high-performance supercomputing facility [30].

First, raw signal data from the Parkes Radio Telescope are recorded at a rate of one gigabyte per second by the ATNF [7] Parkes Swinburne Recorder (APSR) [6]. Depending on the different areas in the universe in which the scientists want to conduct the pulsar searching survey, the observation time is normally from 4 min to 1 h. 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 1–20 GB each in size, depending on the observation time. The beam files contain the pulsar signals which are dispersed by the interstellar medium. De-dispersion is used to counteract this effect. Since the potential dispersion source is unknown, a large number of de-dispersion files needs to 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 takes 1–13 h to finish, and it generates up to 90 GB of de-dispersion files. Furthermore, for binary pulsar searching, every de-dispersion file needs another step of processing named accelerate. This step generates accelerated de-dispersion files with a 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 takes more than one hour to seek the 1200 de-dispersion files. A candidate list of pulsars is generated after the seeking step, which is saved in a text file. Furthermore, by comparing the candidates generated from different beam files in the same time session, some interferences may be detected and some candidates may be eliminated. With the final pulsar candidates, we need to go back to the de-dispersion files to find their feature signals and fold them to XML files. Finally, the XML files are visually displayed to the scientists, 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 needs a long execution time, and large datasets are generated. At present, all the generated datasets are deleted after having been used, and the scientists only store the raw beam data extracted from the raw telescope data. Whenever there are needs to use the deleted datasets, the scientists will regenerate them based on the raw beam files. The generated datasets are not stored, mainly because the supercomputer is a shared facility that cannot offer unlimited storage capacity to hold the accumulated terabytes of data. However, it would be better if some datasets were to be stored, for example, the de-dispersion files, which are frequently used. Based on them, the scientists can apply different seeking algorithms to find potential pulsar candidates. Furthermore, some datasets are derived from the de-dispersion files, such as the results of the seek algorithms and the pulsar candidate list. If these datasets need to be regenerated, the de-dispersion files will also be reused. For large input beam files, the regeneration of the de-dispersion files will take more than 10 h. This not only delays the scientists from conducting their experiments, but also requires a lot of computation resources. On the other hand, some datasets need not 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 used for 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 datasets, they are not worth storing, as it would be more cost effective to regenerate them from the de-dispersion files whenever they are needed.

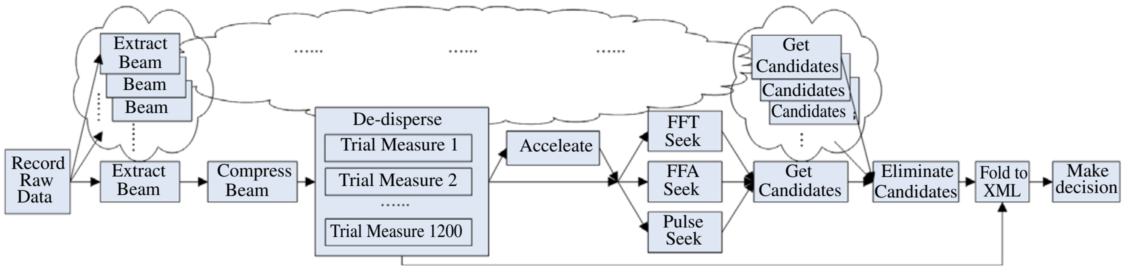


Fig. 1. Pulsar searching workflow.

2.2. Problem analysis

Traditionally, scientific workflows are deployed on high-performance computing facilities, such as clusters and grids. Scientific workflows are often complex, with huge intermediate datasets generated during their execution. How to store these intermediate datasets is normally decided by the scientists who use the scientific workflows. This is because the clusters and grids only serve certain institutions. The scientists may store the intermediate datasets 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 introduced above. The scientists have to delete all the intermediate datasets because of storage limitation. This bottleneck of storage can be avoided if we run scientific workflows in a cloud.

In a cloud computing environment, theoretically, the system can offer unlimited storage resources. All the intermediate datasets 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 datasets or not is no longer an easy decision, for the reasons below.

(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 costs and usage rates. Some of them may be used frequently whilst some others may not. On the one hand, it is most likely not cost effective to store all the intermediate datasets in the cloud. On the other hand, if we delete them all, regeneration of frequently used intermediate datasets imposes a high computation cost. Different storage strategies can be applied to scientific cloud workflow systems, and they will lead to different costs. Traditionally, intermediate dataset storage strategies may be developed based on different factors, such as security, users' preference, etc., but in scientific cloud workflow systems, all the strategies should deem system cost as an important factor. Hence a benchmarking of the minimum system cost is needed to evaluate the cost effectiveness of different intermediate dataset storage strategies.

(2) The usages of intermediate datasets are dynamic. For a single research group, scientists can estimate the intermediate datasets' usages, since they are the only users of the datasets. But for the datasets that are shared among different institutions, their usages are hard to predict. In a cloud computing environment, the users could be anyone from the Internet, and a shared dataset may be used by many users. Hence the datasets' usages are dynamic in scientific cloud workflow systems, and the minimum cost of the system is also a dynamic value. The minimum cost benchmarking should be based on the datasets' usages that are determinate by all the cloud users, and hence should be discovered and obtained from the system logs.

Hence, we need an algorithm to find the minimum cost storage strategy for intermediate datasets in scientific cloud workflow systems based on the historic usages of the datasets. This strategy can be used as a benchmark to evaluate the cost effectiveness over other intermediate dataset storage strategies.

3. Concepts and model of cost-oriented intermediate dataset storage in scientific cloud workflows

In this section, based on our prior work [36,38], we introduce some important concepts, and enhance the representation of the IDG and the dataset storage cost model in scientific cloud workflow systems.

3.1. Classification of the application data in scientific cloud workflows

In general, there are two types of data stored in 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 cannot be regenerated by the system.

Second, *intermediate data* are the data newly generated in the cloud system while the applications run. These data save the intermediate computation results of the applications which will be used in 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 to the result data, they become intermediate data. Therefore, intermediate data are the data generated by computations 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 cannot be regenerated once deleted. For the intermediate data, their storage status can be decided by the system, since they can be regenerated. Hence, our minimum cost storage strategy only applies to intermediate data in scientific cloud workflow systems. In this paper, we refer to intermediate data as dataset(s).

3.2. Data provenance and the intermediate data dependency graph (IDG)

Scientific workflows have many computation and data intensive tasks that generate many intermediate datasets of considerable size. There exist dependencies among the intermediate datasets. Data provenance in workflows is a kind of important metadata in which the dependencies between datasets are recorded [29]. 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 [10]. 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

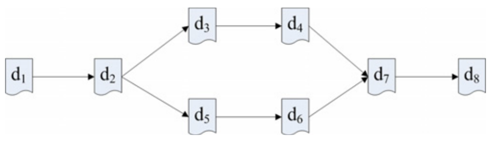


Fig. 2. A simple intermediate data dependency graph (IDG).

from the input data may be very time consuming, and may therefore carry a high cost. In contrast, with data provenance information, the regeneration of the demanding dataset may start from some stored intermediated datasets. In a scientific cloud workflow system, data provenance is recorded during the workflow execution. Taking the advantage of data provenance, we can build the IDG. For all the intermediate datasets, once generated (or modified) in the system, whether stored or deleted, their references are recorded in the IDG as different nodes.

In the IDG, every node denotes an intermediate dataset. Fig. 2 shows us a simple IDG: dataset d_1 pointing to d_2 means that d_1 is used to generate d_2; and d_2 pointing to d_3 and d_5 means that d_2 is used to generate d_3 and d_5 based on different operations; datasets d_4 and d_6 pointing to d_7 means that d_4 and d_6 are used together to generate d_7.

The IDG is a directed acyclic graph (DAG), where no circles exist. This is because the IDG records the provenances of how datasets are derived in the system as time goes on. In other words, it depicts the generation relationships of the datasets.

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 required dataset, so they can be regenerated from their nearest stored predecessors.

We denote a dataset d_i in the IDG as $d_i \in$ IDG, and a set of datasets $S = \{d_1, d_2, \ldots, d_h\}$ in the IDG as $S \subseteq$ IDG. To better describe the relationships of datasets in the IDG, we define two symbols, $\rightarrow$ and $\nleftrightarrow$.

$\rightarrow$ denotes that two datasets have a generation relationship, where $d_i \rightarrow d_j$ means that d_i is a predecessor dataset of d_j in the IDG. For example, in the IDG in Fig. 2, we have $d_1 \rightarrow d_2, d_1 \rightarrow d_4, d_5 \rightarrow d_7, d_1 \rightarrow d_7$, etc. Furthermore, $\rightarrow$ is transitive, where

$$d_i \rightarrow d_j \rightarrow d_k \Leftrightarrow d_i \rightarrow d_j \wedge d_j \rightarrow d_k \Rightarrow d_i \rightarrow d_k.$$

$\nleftrightarrow$ denotes that two datasets do not have a generation relationship, where $d_i \nleftrightarrow d_j$ means the d_i and d_j are in different branches in the IDG. For example, in the IDG in Fig. 2, we have $d_3 \nleftrightarrow d_5, d_3 \nleftrightarrow d_6$, etc. Furthermore, $\nleftrightarrow$ is commutative, where $d_i \nleftrightarrow d_j \Leftrightarrow d_j \nleftrightarrow d_i$.

3.3. Dataset storage cost model

With the IDG, given any intermediate dataset that ever existed in the system, we know how to regenerate it. In this study, we aim at minimising the total cost of managing the intermediate datasets. In a cloud computing environment, if 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 their users. In general, there are two basic types of resource in cloud computing: storage and computation. Popular cloud service providers' cost models are based on these two types of resource [3]. For example, Amazon cloud services' prices are as follows.

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

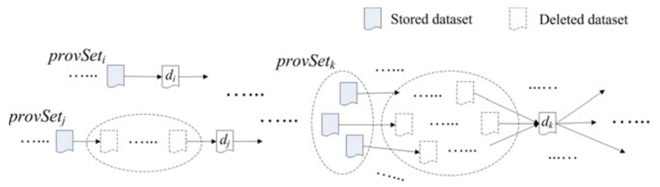


Fig. 3. A dataset's *provSets* in a general IDG.

Furthermore, the cost of data transfer is also considered, such as in Amazon clouds' cost model. In [16], 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 datasets 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 datasets, and S, which is the total cost of storage resources used to store the intermediate datasets.

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 $\langle x_i, y_i, f_i, t_i, provSet_i, CostR_i\rangle$, where the variables have the following meaning.

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 to generate dataset d_i by the price of the computation resources. Normally the generating time can be obtained from the system logs.

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 by the price of the storage resource.

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_i can be defined by the scientists who use this workflow collaboratively. However, a scientific cloud workflow system is based on the Internet, with a large number of users; as we discussed before, t_i cannot be defined by the 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 other words, 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. Fig. 3 shows the *provSet* of a dataset in different situations.

Formally, we can describe a dataset d_i's $ProvSet_i$ as follows:

$$\begin{aligned} provSet_i = \Big\{ d_j \Big| d_j \in \text{IDG} \wedge f_j = \text{“stored”} \wedge d_j \rightarrow d_i \\ \wedge \Big(\big(\neg \exists d_k \in \text{IDG} \wedge d_j \rightarrow d_k \rightarrow d_i\big) \\ \vee \big(\exists d_k \in \text{IDG} \wedge d_j \rightarrow d_k \rightarrow d_i \wedge f_k = \text{“deleted”}\big)\Big)\Big\}. \end{aligned}$$

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

$$genCost(d_i) = x_i + \sum_{\{d_k | d_j \in provSet_i \wedge d_j \rightarrow d_k \rightarrow d_i\}} x_k. \tag{1}$$

This cost is the 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, then, 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 impact on their cost rates. 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 datasets is the sum of $CostR$ of all the intermediate datasets, which is $\sum_{d_i \in \text{IDG}} CostR_i$. We further define the storage strategy of an IDG as S, where $S \subseteq$ IDG, which means storing the datasets in S in the cloud and deleting the others. We denote the cost rate of storing an IDG with the storage strategy S as $\left(\sum_{d_i \in \text{IDG}} CostR_i\right)_S$. 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

$$\text{Total\_Cost} = \int_{t=T_0}^{T_n} \left(\sum_{d_i \in \text{IDG}} CostR_i \right) \bullet \mathrm{d}t. \tag{2}$$

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

4. Minimum cost benchmarking of intermediate dataset storage

The cost rate of scientific applications in the cloud is dynamic. 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 dataset storage strategies. However, based on datasets' usage rates derived from system logs at the time, there exists a minimum cost rate of storing them, which can be used for on-demand benchmarking. In this section, we describe the design of a cost transitive tournament shortest path (CTT-SP) based algorithm that can find the minimum cost storage strategy for a given IDG. The basic idea of the CTT-SP algorithm is to construct a cost transitive tournament (CTT) based on the IDG. In a CTT, we guarantee that the paths from the start dataset to the end dataset have a one-to-one mapping to the storage strategies, and the length of every path equals the total cost rate. Then we can use the well-known Dijkstra algorithm to find the shortest path in the CTT, which is also the minimum cost storage strategy. To describe the algorithm, we start with calculation of the minimum cost benchmark for the linear IDG, and then expand it to the general complex IDG, followed by algorithm complexity analysis.

4.1. Minimum cost algorithm for linear IDG

A linear IDG means an IDG with no branches, where each dataset in the IDG only has one direct predecessor and successor except the first and last datasets.

Take a linear IDG, which has datasets $d_1, d_2, \ldots, d_n$. The CTT-SP algorithm has the following four steps.

Step 1: We add two virtual datasets in the IDG, d_s before d_1 and d_e after d_n, as the start and end datasets, and set $x_s = y_s = 0$ and $x_e = y_e = 0$.

Step 2: We add new directed edges in the IDG to construct the transitive tournament. For every dataset in the IDG, we add edges that start from it and point to all its successors. Formally, for dataset d_i, it has out-edges to all the datasets in the set of $\left\{d_j | d_j \in \text{IDG} \wedge d_i \to d_j\right\}$, and in-edges from all the datasets in the set of $\{d_k | d_k \in \text{IDG} \wedge d_k \to d_i\}$. Hence, for any two datasets d_i and d_j in the IDG, we have an edge between them, denoted as $e\langle d_i, d_j\rangle$. Formally, $d_i, d_j \in \text{IDG} \wedge d_i \to d \Rightarrow \exists e\langle d_i, d_j\rangle$.

Step 3: We set weights to the edges. The reason we call the graph a cost transitive tournament is because the weights of its edges are composed of the cost rates of datasets. For an edge $e\langle d_i, d_j\rangle$, we denote its weight as $\omega\langle d_i, d_j\rangle$, 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,

$$\omega\langle d_i, d_j\rangle = y_j + \sum_{\{d_k | d_k \in \text{IDG} \wedge d_i \to d_k \to d_j\}} (genCost(d_k)/t_k)\,. \tag{3}$$

Since we are discussing a linear IDG, for the datasets between d_i and d_j, d_i is the only dataset in their *provSet*s. Hence we can further get

$$\omega\langle d_i, d_j\rangle = y_j + \sum_{\{d_k | d_k \in \text{IDG} \wedge d_i \to d_k \to d_j\}} \left(\left(x_k + \sum_{\{d_h | d_h \in \text{IDG} \wedge d_i \to d_h \to d_k\}} x_h \right) \Big/ t_k \right). \tag{4}$$

In Fig. 4, we demonstrate a simple example of constructing a CTT for an IDG that only has three datasets, where d_s is the start dataset that only has out-edges and d_e is the end dataset that only has in-edges.

Step 4: We find the shortest path of the CTT. From the construction steps, we can clearly see that the CTT is an acyclic complete oriented graph. Hence we can use the Dijkstra algorithm to find the shortest path from d_s to d_e. The Dijkstra algorithm is a classic greedy algorithm to find the shortest path in graph theory. We denote the shortest path from d_s to d_e as $P_{\min}\langle d_s, d_e\rangle$.

Theorem 1. *Given a linear IDG with datasets $\{d_1, d_2, \ldots, d_n\}$, the length of $P_{\min}\langle d_s, d_e\rangle$ of its CTT is the minimum cost rate of the system to store the datasets in the IDG, and the corresponding storage strategy is to store the datasets that $P_{\min}\langle d_s, d_e\rangle$ traverses.*

Proof. First, there is a one-to-one mapping between the storage strategies of the IDG and the paths from d_s to d_e in the CTT. Given any storage strategy of the IDG, we can find an order of these stored datasets, since the IDG is linear. Then we can find the exact path in the CTT that has traversed all these stored datasets. Similarly, given any path from d_s to d_e in the CTT, we can find the datasets it has traversed, which is a storage strategy. Second, based on the setting of weights to the edges, the length of a path from d_s to d_e in the CTT is equal to the total cost rate of the corresponding storage strategy. Third, $P_{\min}\langle d_s, d_e\rangle$ is the shortest path from d_s to d_e as found by the Dijkstra algorithm. *Hence,* Theorem 1 *holds.* □

Corollary 1. *During the process of finding the shortest path, for every dataset d_f that is discovered by the Dijkstra algorithm, we have a path $P_{\min}\langle d_s, d_f\rangle$ from d_s to d_f and a set of datasets S_f that $P_{\min}\langle d_s, d_f\rangle$ traverses. S_f is the minimum cost storage strategy of the sub-IDG $\left\{d_i | d_i \in \text{IDG} \wedge d_s \to d_i \to d_f\right\}$.*

Proof. By apagoge.

Suppose that there exists a storage strategy $S_f' \neq S_f$, and that S_f' is the minimum cost storage strategy of the sub-IDG

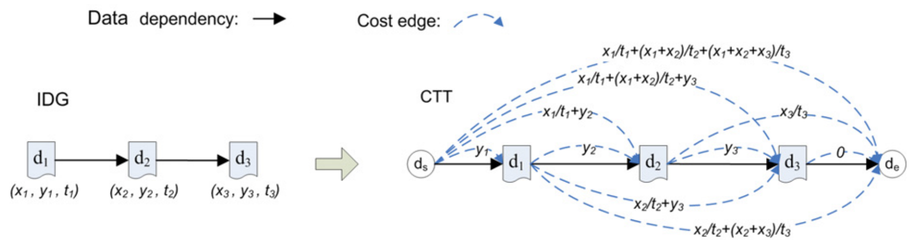
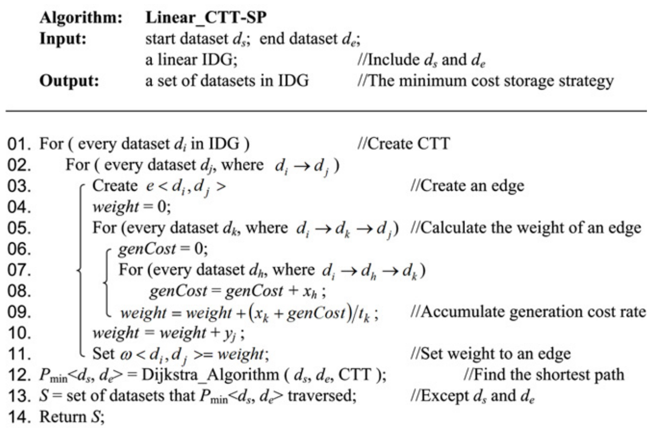


Fig. 4. An example of constructing a CTT.



```
Algorithm:  Linear_CTT-SP
Input:      start dataset $d_s$;  end dataset $d_e$;
            a linear IDG;                   //Include $d_s$ and $d_e$
Output:     a set of datasets in IDG        //The minimum cost storage strategy

01. For ( every dataset $d_i$ in IDG )                     //Create CTT
02.     For ( every dataset $d_j$, where $d_i \to d_j$ )
03.         Create $e < d_i, d_j >$                        //Create an edge
04.         $weight = 0$;
05.         For (every dataset $d_k$, where $d_i \to d_k \to d_j$)  //Calculate the weight of an edge
06.             $genCost = 0$;
07.             For (every dataset $d_h$, where $d_i \to d_h \to d_k$)
08.                 $genCost = genCost + x_h$;
09.             $weight = weight + (x_k + genCost)/t_k$;   //Accumulate generation cost rate
10.         $weight = weight + y_j$;
11.         Set $\omega < d_i, d_j > = weight$;            //Set weight to an edge
12. $P_{\min}<d_s, d_e>$ = Dijkstra_Algorithm ( $d_s$, $d_e$, CTT );   //Find the shortest path
13. $S$ = set of datasets that $P_{\min}<d_s, d_e>$ traversed;    //Except $d_s$ and $d_e$
14. Return $S$;
```

Fig. 5. Pseudo-code of the linear CTT-SP algorithm.

$\{d_i \,|d_i \in \text{IDG} \wedge d_s \to d_i \to d_f\,\}$. Then we can get a path $P'_{\min}\langle d_s, d_f\rangle$ from d_s to d_f, which traverses the datasets in S'_f. Then we have

$$P'_{\min}\langle d_s, d_f\rangle = \left(\sum_{d_i \in \text{IDG} \wedge d_s \to d_i \to d_f} CostR_i\right)_{S'_f}$$

$$< \left(\sum_{d_i \in \text{IDG} \wedge d_s \to d_i \to d_f} CostR_i\right)_{S_f} = P_{\min}\langle d_s, d_f\rangle.$$

This is contradictory to the known condition "$P_{\min}\langle d_s, d_f\rangle$ is the shortest path from d_s to d_f". Hence, S_f is the minimum cost storage strategy of the sub-IDG $\{d_i \,|d_i \in \text{IDG} \wedge d_s \to d_i \to d_f\,\}$. *Hence,* Corollary 1 *holds.* □

Fig. 5 shows the pseudo-code of the linear CTT-SP algorithm. To construct the CTT, we first create the cost edges (line 3), and then calculate their weights (lines 4–11). Next, we use the Dijkstra algorithm to find the shortest path (line 12), and return the minimum cost storage strategy (lines 13–14).

4.2. Minimum cost algorithm for an IDG with one block

A linear IDG is a special case of general IDGs. In the real world, intermediate datasets generated in scientific workflows may have complex relationships, such that different datasets may be generated from a single dataset by different operations, and different datasets may be used together to generate one dataset. In other words, the IDG may have branches, where the linear CTT-SP algorithm introduced in Section 4.1 cannot be directly applied. This is because the CTT can only be constructed on a linear IDG, which means that the datasets in the IDG must be totally ordered. In this section, we discuss how to find the minimum cost storage strategy for an IDG that has a sub-branch within one block.

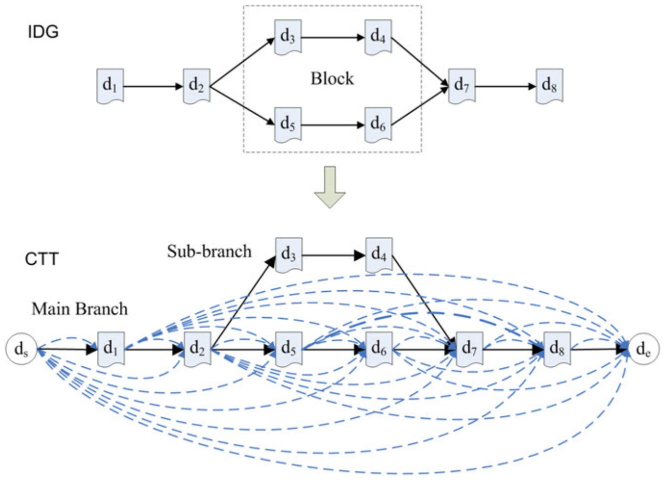


Fig. 6. An example of constructing the CTT for an IDG with a block.

4.2.1. Construct the CTT for an IDG with a block

First we introduce the concept of a "block" in an IDG. A block is a set of sub-branches in the IDG that split from a common dataset and merge into another common dataset. We denote the block as B. Fig. 6 shows an IDG with a simple block $B = \{d_3, d_4, d_5, d_6\}$; we will use it as the example to illustrate the construction of the CTT.

To construct the CTT, we need the datasets in the IDG to be totally ordered. Hence, for an IDG with a block, we only choose one branch to construct the CTT, as shown in Fig. 6. We call the linear datasets which are chosen to construct the CTT the "main branch", denoted as MB, and call the rest of the datasets "sub-branches", denoted as SB. For example, in the IDG in Fig. 6, $MB = \{d_1, d_2, d_5, d_6, d_7, d_8\}$ and $SB = \{d_3, d_4\}$. Due to the existence of the block, the edges can be classified into four categories. The definitions of this classification are as follows.

- *In-block edge*: $e\langle d_i, d_j\rangle$ 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 $e\langle d_2, d_5\rangle$, $e\langle d_1, d_6\rangle$ in Fig. 6. Formally, we define $e\langle d_i, d_j\rangle$ as an in-block edge, where $\exists d_k \in \text{IDG} \wedge d_i \to d_k \wedge d_j \nleftrightarrow d_k$.
- *Out-block edge*: $e\langle d_i, d_j\rangle$ 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 $e\langle d_6, d_7\rangle$, $e\langle d_5, d_8\rangle$ in Fig. 6. Formally, we define $e\langle d_i, d_j\rangle$ as an out-block edge, where $\exists d_k \in \text{IDG} \wedge d_i \nleftrightarrow d_k \wedge d_k \to d_j$.
- *Over-block edge*: $e\langle d_i, d_j\rangle$ is an over-block edge means that the edge crosses over the block, where d_i is a dataset preceding the block and d_j is a dataset succeeding the block, such as $e\langle d_2, d_7\rangle$, $e\langle d_1, d_8\rangle$ in Fig. 6. Formally, we define $e\langle d_i, d_j\rangle$ as an over-block edge, where $\exists d_k, d_h \in \text{IDG} \wedge d_h \nleftrightarrow d_k \wedge d_i \to d_h \to d_j \wedge d_i \to d_k \to d_j$.
- *Ordinary edge*: $e\langle d_i, d_j\rangle$ is an ordinary edge means that datasets between d_i and d_j are totally ordered, such as $e\langle d_s, d_2\rangle$,

$e\langle d_5, d_6\rangle$, $e\langle d_7, d_8\rangle$ in Fig. 6. Formally, we define $e\langle d_i, d_j\rangle$ as an ordinary edge, where

$$\neg\exists d_k \in \text{IDG} \wedge ((d_i \rightarrow d_k \wedge d_k \nleftrightarrow d_j) \vee (d_i \nleftrightarrow d_k \wedge d_k \rightarrow d_j) \vee (d_h \in \text{IDG} \wedge d_h \nleftrightarrow d_k \wedge d_i \rightarrow d_h \rightarrow d_j \wedge d_i \rightarrow d_k \rightarrow d_j)).$$

4.2.2. Setting weights to different types of edges

The essence of the CTT-SP algorithm is the rules of setting weights to the cost edges. Based on the setting, we guarantee that the paths from the start dataset d_s to every dataset d_i in the IDG represent the storage strategies of the datasets $\{d_k | d_k \in \text{IDG} \wedge d_s \rightarrow d_k \rightarrow d_i\}$, and the shortest path is the minimum cost storage strategy. As defined in Section 4.1, the weight of edge $e\langle d_i, d_j\rangle$ is the sum of the 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 the rest of the datasets between d_i and d_j are all deleted. In an IDG with one block, this rule is still applicable to the ordinary edges and in-block edges.

However, if $e\langle d_i, d_j\rangle$ is an out-block edge or over-block edge, formula (3) in Section 4.1 is not applicable for calculating its weight anymore, for the following reasons.

(1) Due to the existence of the block, the datasets succeeding the block may have more than one dataset in their *provSet*s. The generation of these datasets needs not only d_i, but also the stored provenance datasets from the other sub-branches of the block. For example, according to formula (3) in Section 4.1, the weight of out-block edge $e\langle d_5, d_8\rangle$ in Fig. 6 is $\omega\langle d_5, d_8\rangle = y_8 + genCost(d_6)/t_6 + genCost(d_7)/t_7$, where if we want to calculate $genCost\ (d_7)$, we also have to know the storage statuses of d_3 and d_4. The same problem also exists when calculating the weights of the over-block edges. Hence, to calculate the weights of the out-block and over-block edges, we have to know the storage strategies of all the sub-branches in the block.

(2) The path from d_s to d_j cannot represent the storage strategy of all the datasets $\left\{d_k \middle| d_k \in \text{IDG} \wedge d_s \rightarrow d_k \rightarrow d_j\right\}$. If we use the same method as in Section 4.1 to set the weight of $e\langle d_i, d_j\rangle$, the path that contains $e\langle d_i, d_j\rangle$ in the CTT can only represent the storage strategy of datasets in the main branch, where the sub-branches are not represented. For example, in Fig. 6, the path from d_s to d_8 that contains out-block edge $e\langle d_5, d_8\rangle$ does not represent the storage statuses of datasets d_3 and d_4, and the length of the path also does not contain the cost rates of d_3 and d_4, if we use the method in Section 4.1 to calculate the weights of the edges. Hence, to maintain the mapping between the paths and the storage strategies, the weights of the out-block and over-block edges should contain the minimum cost rates of the datasets in the sub-branches of the block.

Based on the reasons above, we define the weight of $e\langle d_i, d_j\rangle$ as

$$\omega\langle d_i, d_j\rangle = y_j + \sum_{\{d_k | d_k \in MB \wedge d_i \rightarrow d_k \rightarrow d_j\}} (genCost(d_k)/t_k) + \left(\sum_{\{d_h | d_h \in SB\}} CostR_h\right)_{S_{\min}}. \quad (5)$$

In formula (5), $\left(\sum_{\{d_h | d_h \in SB\}} CostR_h\right)_{S_{\min}}$ means the minimum cost rates of the datasets that are in the sub-branches of the block. This formula guarantees that the length of the shortest path with an out-block edge or over-block edge still equals the minimum cost rate of the datasets, which is $P_{\min}\langle d_s, d_j\rangle = \left(\sum_{\{d_k | d_k \in \text{IDG} \wedge d_s \rightarrow d_k \rightarrow d_j\}} CostR_k\right)_{S_{\min}}$. Hence, to calculate the weights of the out-block and over-block edges, we have to find the minimum cost storage strategy of the datasets that are in the sub-branches of the block. For example, the weight of edge $e\langle d_5, d_8\rangle$ in Fig. 6 is $\omega\langle d_5, d_8\rangle = y_8 + genCost(d_6)/t_6 + genCost(d_7)/t_7 + (CostR_3 + CostR_4)_{S_{\min}}$, where we have to find the minimum cost storage strategy of datasets d_3 and d_4.

However, for any sub-branches, the minimum cost storage strategy is dependent on the storage status of the datasets preceding and succeeding the block (i.e. stored adjacent predecessor and successor of the sub-branches).

If $e\langle d_i, d_j\rangle$ is an over-block edge, according to its semantics, d_i and d_j are stored datasets, and the datasets between d_i and d_j in the main branch, $\left\{d_k \middle| d_k \in MB \wedge d_i \rightarrow d_k \rightarrow d_j\right\}$, are deleted. Hence, d_i and d_j are the stored adjacent predecessor and successor of the sub-branches. If the remaining datasets within the block form a linear IDG, we can use the linear CTT-SP algorithm introduced in Section 4.1 to find its minimum cost storage strategy, where in the first step we have to use d_i and d_j as the start and end datasets. For example, to calculate the weight of the over-block edge $e\langle d_1, d_8\rangle$ in Fig. 6, we have to find the minimum cost storage strategy of sub-branch $\{d_3, d_4\}$ by the linear CTT-SP algorithm, given that d_1 is the start dataset and d_8 is the end dataset. Otherwise, if the remaining datasets within the block do not form a linear IDG, we have to recursively call the CTT-SP algorithm to find the minimum cost storage strategy of the sub-branches, which will be introduced in Section 4.3. Hence, the weight of an over-block edge can be calculated.

If $e\langle d_i, d_j\rangle$ is an out-block edge, we only know that the stored adjacent successor of the sub-branches is d_j. However, the minimum cost storage strategy of the sub-branches is also dependent on the stored adjacent predecessor, which is unknown for an out-block edge. Hence, given different stored adjacent predecessors, the weight of an out-block edge would be different. For example, to calculate the weight of out-block edge $e\langle d_5, d_8\rangle$ in Fig. 6, we need to find the minimum cost storage strategy $S_{\min}$ of the sub-branch $\{d_3, d_4\}$, where we only know the stored adjacent successor d_8. However, $S_{\min}$ may be different depending on the storage statuses of d_1 and d_2. Hence, we have to create multiple CTTs for an IDG that has a block, in order to calculate the weights of the out-block edges in different situations, as detailed next.

4.2.3. Steps of finding the minimum cost storage strategy for an IDG with one sub-branch in the block

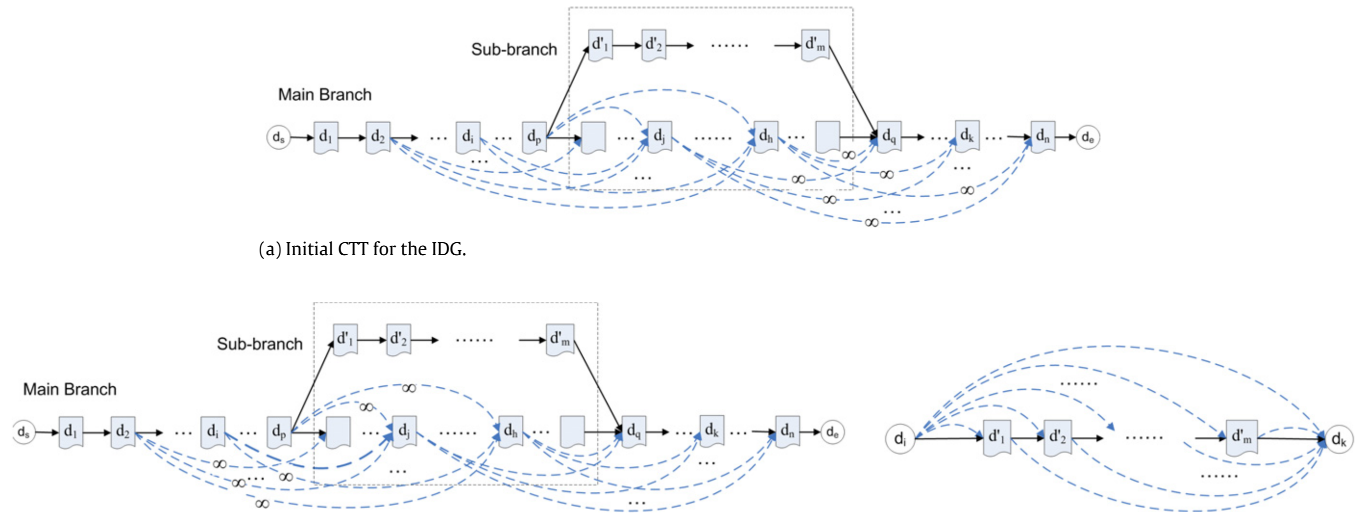
In this section, we extend the linear CTT-SP algorithm to find its minimum cost storage strategy for an IDG with one sub-branch in the block. As discussed in Section 4.2.2, depending on different stored preceding datasets of the block, the weight of an out-block edge may be different. Hence multiple CTTs are needed to represent these different situations, and the minimum cost storage strategy is the shortest path among all the CTTs.

To develop the minimum cost storage strategy, we need the following two theorems.

Theorem 2. *The selection of the main branch in the IDG to construct the CTT has no impact on finding the minimum cost storage strategy.*

Proof. Assume that strategy S is the minimum cost storage strategy of an IDG; the IDG has two sub-branches Br_1 and Br_2 in a block; strategies S_1 and S_2 contain the sets of stored datasets of Br_1 and Br_2 in S.

If we select the main branch with the sub-branch Br_1, S can be mapped to a path in one of the created CTTs. According to Theorem 1, the paths in the CTT have one-to-one mapping to the storage strategies; hence we can find a path $P\langle d_s, d_e\rangle$ that traverses the stored datasets in the main branch according to S. If $S_1 = \emptyset$, there is an over-block edge in the path $P\langle d_s, d_e\rangle$, which contains the minimum cost storage strategy of Br_2 according to formula (5), where $P\langle d_s, d_e\rangle$ is in the initial CTT. If $S_1 \neq \emptyset$, there is an in-block edge and an out-block edge in $P\langle d_s, d_e\rangle$, denoted as $e\langle d_i, d_j\rangle$ and $e\langle d_h, d_k\rangle$. The



(a) Initial CTT for the IDG.

(b) CTT($e\langle d_i, d_j\rangle$) for in-block edge $e\langle d_i, d_j\rangle$.

(c) CTT created for the sub-branch.

Fig. 7. CTTs for an IDG with one block.

weight of $e\langle d_h, d_k\rangle$ contains the minimum cost storage strategy of Br_2 according to formula (5); hence $P\langle d_s, d_e\rangle$ is in CTT ($e\langle d_i, d_j\rangle$). Similar to Theorem 1, we can further prove that the length of $P\langle d_s, d_e\rangle$ equals the total cost rate of the storage strategy S.

Similarly, if we select the main branch with the sub-branch Br_2, S can also be mapped to a path in one of the created CTTs, where the length of the path is equal to the total cost rate of the minimum cost storage strategy.

Therefore, no matter which branch we select as the main branch to construct the CTT, the minimum cost storage strategy always exists in one of the created CTTs. This means that the selection of the main branch has no impact on finding the minimum cost storage strategy. *Hence*, Theorem 2 *holds.* □

Theorem 3. *The Dijkstra algorithm is still applicable for finding the minimum cost storage strategy of the IDG with one block.*

Proof. In the CTTs created for an IDG with one block, every path from d_s to d_e contains an out-block edge or over-block edge. According to formula (5), the minimum cost rate of the sub-branch is contained in the weights of the out-block and over-block edges. Hence, every path from d_s to d_e in the CTT contains the minimum cost storage strategy of the sub-branch. Furthermore, the CTTs are created based on the main branch of the IDG; similar to the proof of Theorem 1, the shortest path $P_{\min}\langle d_s, d_e\rangle$ found by the Dijkstra algorithm contains the minimum cost storage strategy of the main branch. This means that $P_{\min}\langle d_s, d_e\rangle$ represents the minimum cost storage strategy of the whole IDG. *Hence*, Theorem 3 *holds.* □

The main steps of the algorithm are as follows.

Step 1: Construct the initial CTT of the IDG. According to Theorem 2, we choose an arbitrary branch in the IDG as the main branch and add cost edges to construct the CTT. In the CTT, for the ordinary edges and in-block edges, we set their weights based on formula (3) in Section 4.1. For the over-block edges, we set their weights according to formula (5) by calling the linear CTT-SP algorithm to find the minimum cost storage strategy of the sub-branch, which is introduced in Section 4.2.2. For the out-block edges, we set their weights as infinity at the initial stage. The initial CTT is shown in Fig. 7(a).

Step 2: Based on Theorem 3, start the Dijkstra algorithm to find the shortest path from d_s to d_e. We use F to denote the set of datasets discovered by the Dijkstra algorithm. When a new edge $e\langle d_i, d_j\rangle$ is discovered, we first add d_j to F, and then check whether $e\langle d_i, d_j\rangle$ is an in-block edge or not. If not, we continue to find the next edge by the Dijkstra algorithm until d_e is reached which would terminate the algorithm. If $e\langle d_i, d_j\rangle$ is an in-block edge, create a new CTT (see Steps 2.1–2.3) because whenever an in-block edge is discovered, a stored adjacent predecessor of the sub-branch is identified, and this dataset will be used in calculating the weights of the out-block edges.

Step 2.1: In the case where in-block edge $e\langle d_i, d_j\rangle$ is discovered, based on the current CTT, create CTT($e\langle d_i, d_j\rangle$), as shown in Fig. 7(b). First, we copy all the information of the current CTT to the new CTT($e\langle d_i, d_j\rangle$). Second, we update the weights of all the in-block edges in CTT($e\langle d_i, d_j\rangle$) as infinity, except $e\langle d_i, d_j\rangle$. This guarantees that dataset d_i is the stored adjacent predecessor of the sub-branch in all the paths of CTT($e\langle d_i, d_j\rangle$). Third, we update the weights of all the out-block edges in CTT($e\langle d_i, d_j\rangle$) as described in Step 4 below.

Step 2.2: Calculate the weight of an out-block edge $e\langle d_h, d_k\rangle$ in CTT($e\langle d_i, d_j\rangle$). As discussed in Section 4.2.2, to calculate the weight of $e\langle d_h, d_k\rangle$ according to formula (5), we have to find the minimum cost storage strategy of the sub-branch in the block. From Fig. 7(b), we can see that the sub-branch is $\{d'_1, d'_2, \ldots, d'_m\}$, which is a linear IDG. We can find its minimum cost storage strategy by using the linear CTT-SP algorithm described in Section 4.1, given that d_i is the start dataset and d_k is the end dataset. The CTT created for the sub-branch is depicted in Fig. 7(c).

Step 2.3: Add the new CTT($e\langle d_i, d_j\rangle$) to the CTT set.

4.3. Minimum cost algorithm for a general IDG

In real-world applications, the structure of the IDG could be complex, i.e. there may exist more than one block in an IDG. However, to calculate the minimum cost storage strategy of a general IDG, no matter how complex the IDG's structure is, we can reduce the calculation process to linear IDG situations by recursively calling the algorithm introduced in Section 4.2. In this section we introduce the general CTT-SP algorithm as well as the pseudo-code for calculating the minimum cost storage strategy for a general IDG.

4.3.1. General CTT-SP algorithm

The complex structure of an IDG can be viewed as a combination of many blocks. Following the algorithm steps introduced in Section 4.2, we choose an arbitrary branch from the start dataset d_s to the end dataset d_e as the main branch to construct the initial CTT

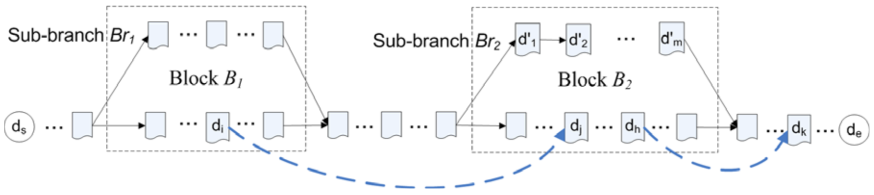


Fig. 8. A sub-branch IDG has more than one stored adjacent predecessor.

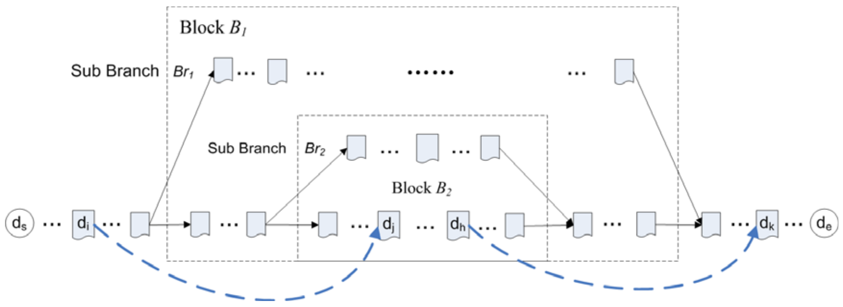


Fig. 9. A sub-branch IDG also has branches.

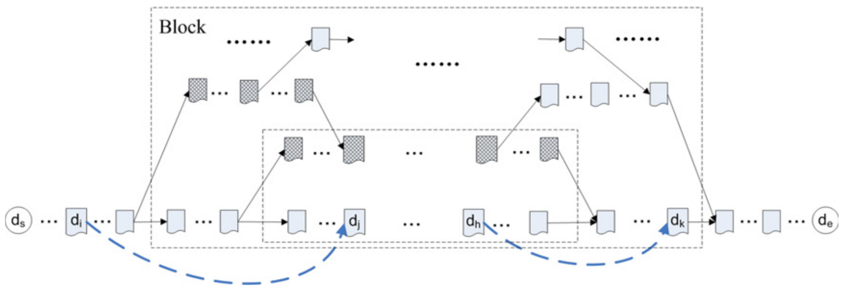


Fig. 10. The CTT for a general IDG.

and create multiple CTTs for the different in-block edges which are discovered by the Dijkstra algorithm. In the process of calculating the weights of the out-block and over-block edges, there are two new situations for finding the minimum cost storage strategy of the sub-branches.

(1) The sub-branches may have more than one stored adjacent predecessors. For example, $e\langle d_i, d_j\rangle$ in Fig. 8 is an out-block edge of block B_1, and also an in-block edge of block B_2. In the algorithm, if edge $e\langle d_i, d_j\rangle$ is found by the Dijkstra algorithm, we create a new CTT($e\langle d_i, d_j\rangle$) from the current CTT, since $e\langle d_i, d_j\rangle$ is an in-block edge of block B_2. To calculate the weights of the out-block edges in CTT($e\langle d_i, d_j\rangle$), for example $e\langle d_h, d_k\rangle$ in Fig. 8, we need to find the minimum cost storage strategy of sub-branch $\{d'_1, d'_2, \ldots, d'_m\}$ of block B_2. However, because $e\langle d_i, d_j\rangle$ is also an out-block edge of B_1, d_i is not the only dataset in d'_1's *provSet*. To calculate the generation cost of d'_1, we need to find its stored provenance datasets from sub-branch Br_1 of block B_1.

(2) The sub-branches are a general IDG which also has branches. In this situation, we need to recursively call the general CTT-SP algorithm to calculate its minimum cost storage strategy. For example, $e\langle d_i, d_j\rangle$ in Fig. 9 is an in-block edge of blocks B_1 and B_2. If $e\langle d_i, d_j\rangle$ is selected by the algorithm, we need to create a new CTT($e\langle d_i, d_j\rangle$). To calculate the weight of $e\langle d_h, d_k\rangle$ in Fig. 9, which is an out-block edge of both B_1 and B_2, we need to calculate the minimum cost storage strategy of sub-branches Br_1 and Br_2. Hence we have to recursively call the general CTT-SP algorithm for the IDG $Br_1 \cup Br_2$, given the start dataset d_i and the end dataset d_k.

Hence, given a general IDG, its structure can be viewed as the combination of many blocks. By recursively calling the general CTT-SP algorithm for the sub-branches, we can eventually find the minimum cost storage strategy of the whole IDG. Fig. 10 shows an example of a general IDG. To create CTT($e\langle d_i, d_j\rangle$), we need to calculate the weights of all the out-block edges. For example, for out-block edge $e\langle d_h, d_k\rangle$, we need to further calculate the minimum cost storage strategy of the sub-branches $\left\{d_u \,\middle|\, d_u \in \text{IDG} \wedge d_u \rightarrow d_k \wedge d_u \nleftrightarrow d_j \wedge d_u \nleftrightarrow d_h\right\}$, as shadowed in Fig. 10, given the start dataset d_i and the end dataset d_k.

4.3.2. Pseudo-code of the general CTT-SP algorithm

Fig. 11 shows the pseudo-code of the general CTT-SP algorithm. First, we choose an arbitrary branch from d_s to d_e as the main branch to construct the initial CTT (lines 1–21), where we need to recursively call the general CTT-SP algorithm in calculating the weights for the over-block edges (lines 11–14). Then we start the Dijkstra algorithm (lines 22–50). Whenever an in-block edge is found, we construct a new CTT with the following steps. First we create a copy of the current CTT, in which the in-block edge is found (line 31). Next, we update the weights of the edges: lines 32–34 are for updating the weights of the in-block edges and lines 35–49 are for updating the weights of the out-block edges. If the sub-branch is a linear IDG, we call the linear CTT-SP algorithm described in Fig. 5; otherwise we recursively call the general CTT-SP algorithm (lines 39–42). Finally, we add the new CTT to the CTTSet (line 50) and continue the Dijkstra algorithm to find the next edge. When the end dataset d_e is reached, the algorithm ends with the minimum cost storage strategy returned.

4.4. Complexity analysis of minimum cost algorithms

From the pseudo-code in Fig. 5, we can clearly see that, for a linear IDG with n datasets, we have to add a magnitude of n^2 edges

```
Algorithm:  General_CTT-SP
Input:      start dataset $d_s$;  end dataset $d_e$;
            a general IDG;                                   //Include $d_s$ and $d_e$
Output:     a set of datasets in the IDG;                    //The minimum cost storage strategy
```

```
Get a main branch $MB$ from IDG;
For ( every dataset $d_i$ in $MB$ )                                   //Create initial CTT
    For ( every dataset $d_j$, where $d_j \in MB \wedge d_i \to d_j$ )
        Create $e < d_i, d_j >$;                                      //Create an edge
        If ( $\exists d_k \in IDG \wedge d_i \not\leftrightarrow d_k \wedge d_k \to d_j$ )   //e is an out-block edge
            Set $\omega < d_i, d_j >= \infty$;
        else                                                          //Calculate the weight of the edge
            $weight = 0$;
            If ( $\exists d_k \notin MB \wedge d_i \to d_k \to d_j$ )   //e is an over-block edge
                $SB = \{ d_k \mid d_k \notin MB \wedge d_i \to d_k \to d_j \}$;   //Get the sub-branches SB
                If (SB is linear)                                     //Find the minimum cost storage strategy of SB
                    S' = Linear_CTT-SP($d_i$, $d_j$, SB);
                else
                    S' = General_CTT-SP($d_i$, $d_j$, SB);
                $weight = weight + \left( \sum_{d_i \in SB} CostR_i \right)_{S'}$;
            For (every dataset $d_k$, where $d_k \in MB \wedge d_i \to d_k \to d_j$)   //Datasets in the main branch
                $genCost = 0$;
                For (every dataset $d_h$, where $d_h \in MB \wedge d_i \to d_h \to d_k$)
                    $genCost = genCost + x_h$;
                $weight = weight + (x_k + genCost)/t_k$;              //Sum of generation cost rates
            Set $\omega < d_i, d_j >= weight + y_j$;                  //Set weight to the edge
$CTTSet = \{CTT_{ini}\}$;                                             //Set of all the created CTTs
$F = \{\emptyset\}$;                                                  //Set of datasets discovered by Dijkstra algorithm
While ( $d_e$ is not in $F$ )
    For ( every $CTT$ in $CTTSet$ )                                   //Find the next edge for the shortest path
        Find the next edge by Dijkstra algorithm;
    Get the current shortest path in all the CTTs, which is with the edge $e < d_i, d_j > \in CTT'$
    Add $d_j$ to $F$;
    If ( $\exists d_b \in IDG \wedge d_i \to d_b \wedge d_j \not\leftrightarrow d_b$ )   //e is an in-block edge
        $BSet = \{ B_p \mid B_p \subset IDG \wedge d_i \notin B_p \wedge d_j \in B_p \}$;   //The blocks that contains $d_j$ but not $d_i$
        Create a copy of CTT' denoted as CTT(e<$d_i$ , $d_j$>);       //Create a new CTT for the in-block edge
        For ( every $B_p \in BSet$)                                   //Update the weights of the in-block edges
            For ( every $e < d_r, d_t > \neq e < d_i, d_j >$ where $d_r \notin B_p \wedge d_t \in B_p$)
                Set $\omega < d_r, d_t >= \infty$;
        For ( every $B_p \in BSet$ )                                  //Update the weights of the out-block edges
            For ( every $e < d_h, d_k >$ where $d_h \in B_p \wedge d_j \to d_h \wedge d_k \notin B_p$)
                $weight = 0$;
                $SB = \{ d_p \mid d_p \in IDG \wedge d_i \to d_p \to d_k \wedge d_p \not\leftrightarrow d_j \wedge d_p \not\leftrightarrow d_h \}$;   //Get the sub-branches SB
                If (SB is linear)                                     //Find the minimum cost storage strategy of SB
                    S' = Linear_CTT-SP($d_i$, $d_k$, SB);
                else
                    S' = General_CTT-SP($d_i$, $d_k$, SB);
                $weight = \left( \sum_{d_i \in SB} CostR_i \right)_{S'}$;
                For (every dataset $d_l$, where $d_l \in MB \wedge d_h \to d_l \to d_k$)   //Datasets in the main branch
                    $genCost = 0$;
                    For (every dataset $d_o$, where $d_o \in MB \wedge d_h \to d_o \to d_l$)
                        $genCost = genCost + x_o$;
                    $weight = weight + (x_l + genCost)/t_l$;          //Sum of generation cost rate
                Set $\omega < d_h, d_k >= weight + y_k$;              //Set weight to the out-block edge
        Add CTT(e<$d_i$ , $d_j$>) to CTTSet;
Return S = set of datasets that the shortest path from $d_s$ to $d_e$ has traversed;
```

Fig. 11. Pseudo-code of the general CTT-SP algorithm.

to construct the CTT (line 3 with two nested loops in lines 1–2), and for the longest edge, the time complexity of calculating its weight is also $O(n^2)$ (lines 5–11 with two nested loops), so we have a total of $O(n^4)$. Next, the Dijkstra algorithm (line 12) has the known time complexity of $O(n^2)$. Hence the linear CTT-SP algorithm has a worst-case time complexity of $O(n^4)$. Furthermore, the space complexity of the linear CTT-SP algorithm is the space of storing the CTT, which is $O(n^2)$.

From the pseudo-code in Fig. 11, we can see that recursive calls (line 14 and line 42) exist in the general CTT-SP algorithm, which makes the algorithm's complexity highly dependent on the structure of the IDG. Next, we analyse the worst-case scenario of the algorithm and show that both the time and space complexities are polynomial.

In Fig. 11, pseudo-code lines 1–21 are for constructing one CTT, i.e. the initial CTT. From pseudo-code lines 24 to 50 of the general CTT-SP algorithm, many CTTs are created for the IDG during the process of the Dijkstra algorithm, which determine the algorithm's computation complexity. The maximum number of created CTTs is smaller than the number of datasets in the main branch, which is in the magnitude of n. Hence, if we denote the time complexity of the general CTT-SP algorithm as $F_l(n)$, we have a recursive equation as follows:

$$\begin{cases} F_0(n) = O(n^4) \\ F_l(n) = n^3 * \left(F_{l-1}(n_{(l-1)}) + n^2\right), \quad l > 0. \end{cases} \tag{6}$$

In Eq. (6), n is the number of datasets in the IDG, $n_{(l-1)}$ is the number of datasets in the sub-branches, and l is the maximum level of the recursive calls; in particular, $F_0(n)$ denotes the situation of a linear IDG, where the linear CTT-SP algorithm needs to be called (i.e. pseudo-code in Fig. 5).

Intuitively, in Eq. (6), $F_l(n)$ seems to have an exponential complexity depending on the level of recursive calls. However, in our scenario, $F_l(n)$ is polynomial because the recursive call is to find the minimum cost storage strategy of given sub-branches in an IDG which has a limited solution space. Hence, we can use the iterative method [26] to solve the recursive equation and derive the computation complexity of the general CTT-SP algorithm.

If we assume that we have already found the minimum cost storage strategies for all sub-branches, this means, without taking the impact of recursive calls into account, that the general CTT-SP algorithm has a time complexity of $O(n^5)$. Formally, we can transform Eq. (6) to the following:

$$\begin{aligned} F_l(n) &= n^3 * \left(O(1) + n^2\right) + f_{\text{rec}}\left(F_{l-1}(n_{(l-1)})\right) \\ &= O(n^5) + f_{\text{rec}}\left(F_{l-1}(n_{(l-1)})\right). \end{aligned} \tag{7}$$

In Eq. (7), function f_{rec} denotes the complexity of recursive calls, i.e. calculating the minimum cost storage strategies of all sub-branches. Next, we analyse the complexity of recursive calls.

For a sub-branch of a general IDG, given different start dataset and end dataset, its minimum cost storage strategy may be different. Fig. 12 shows a sub-branch of an IDG with w datasets. We assume that d_1's direct predecessors and d_w's direct successors are all stored; then we can calculate a minimum cost storage strategy of the sub-branch. We denote the first stored dataset as d_u and the last stored dataset as d_v in the strategy, which is shown in Fig. 12. If d_1's adjacent stored predecessors are changed, the minimum cost storage strategy may be different as well. Because the generation cost of d_1 is larger than that of storing the direct predecessors, the first stored dataset in the new strategy must be one of the datasets from d_1 to d_u. Similarly, if d_w's adjacent stored successors are changed, the last stored dataset in the new strategy must be one of the datasets from d_v to d_w. Hence, given different start and end datasets, a sub-branch of the IDG has at most $u * (w - v)$ different minimum cost storage strategies, which are of the magnitude of

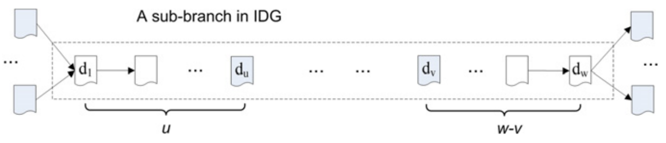


Fig. 12. A sub-branch in the IDG.

w^2. Similarly, we can prove that, for any sub-branches of IDG with w datasets, there are at most w^2 different minimum cost storage strategies, given different start and end datasets. Hence, given any sub-branches in IDG at any level of recursive calls, say level h, we have the time complexity $F_h(w) * w^2$ of finding all the possible minimum cost storage strategies.

If we assume that there are m different sub-branches of recursive calls at level h for which we have to find their minimum cost storage strategies, we have the complexity of recursive calls at this level as follows:

$$f_{\text{rec}}(F_h(n_h)) \le \sum_{i=1}^{m} \left(F_h(n_{h,i}) * n_{h,i}^2\right). \tag{8}$$

With formula (8), we can further transform Eq. (7) and iteratively derive the time complexity of the general CTT-SP algorithm.

Therefore, the entire iteration process from Eq. (6) is shown as follows:

$$\begin{aligned} F_l(n) &= n^3 * \left(F_{l-1}(n_{(l-1)}) + n^2\right) \\ &= O(n^5) + f_{\text{rec}}\left(F_{l-1}(n_{(l-1)})\right) \qquad // \text{ from } (7) \\ &\le O(n^5) + \sum_{i=1}^{m_{l-1}} \left(F_{l-1}(n_{(l-1),i}) * {n_{(l-1),i}}^2\right) \quad // \text{ from } (8) \\ &= O(n^5) + \sum_{i=1}^{m_{l-1}} \left({n_{(l-1),i}}^3 * \left(F_{l-2}(n_{(l-2),i}) \right.\right. \\ &\quad \left.\left. + {n_{(l-1),i}}^2\right) * {n_{(l-1),i}}^2\right) \quad // \text{ recursion} \\ &\le O(n^5) + \sum_{i=1}^{m_{l-1}} \left(O\left((n_{(l-1),i})^5\right) * {n_{(l-1),i}}^2\right) \\ &\quad + \sum_{i=1}^{m_{l-2}} \left(F_{l-2}(n_{(l-2),i}) * {n_{(l-2),i}}^2\right) \quad // \text{ from } (7)(8) \\ &\le O(n^5) + \sum_{i=1}^{m_{l-1}} \left(O\left((n_{(l-1),i})^5\right) * {n_{(l-1),i}}^2\right) + \cdots \\ &\quad + \sum_{i=1}^{m_0} \left(F_0(n_{0,i}) * {n_{0,i}}^2\right) \quad // \text{ iteration} \\ &= O(n^5) + \sum_{j=l-1}^{1} \left(\sum_{i=1}^{m_j} \left(O\left((n_{j,i})^5\right) * {n_{j,i}}^2\right)\right) \\ &\quad + \sum_{i=1}^{m_0} \left(O({n_{0,i}}^4) * {n_{0,i}}^2\right) \quad // \text{ from } F_0(n) = O(n^4) \\ &\le l * m * O(n^5) * n^2 \quad // m = \max_{i=0}^{j}(m_i) \\ &\le O(n^9) \quad // l < n, m < n. \end{aligned}$$

Hence, the worst-case time complexity of the general CTT-SP algorithm is $O(n^9)$.

Similarly, the space complexity of the general CTT-SP algorithm is $l * m * n^2 * O(n^3)$, where the worst case is $O(n^7)$.

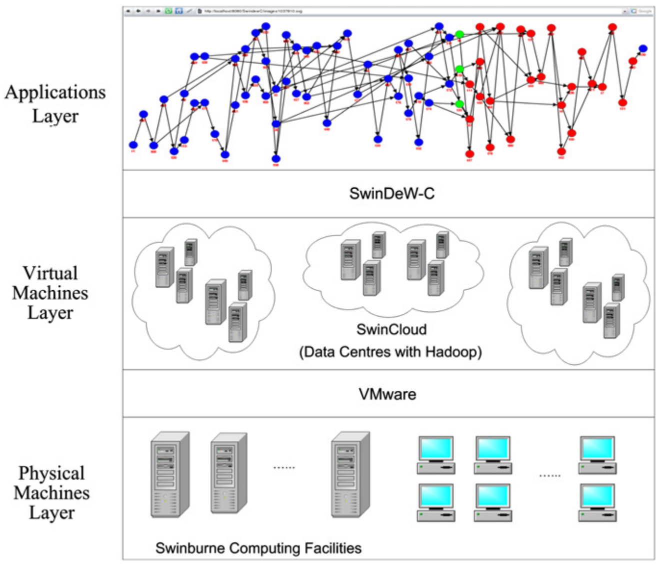


Fig. 13. Structure of the simulation environment.

5. Evaluation

The on-demand minimum cost benchmarking for intermediate dataset storage strategies proposed in this paper is generic. It can be used in any scientific workflow application. In this section, we demonstrate the simulation results that we conduct on the SwinCloud system [23]. We start with a description of the simulation environment and strategies. Then we evaluate general (random) workflows to demonstrate the comparison of our benchmark with different storage strategies. Finally, we use our algorithm with the specific pulsar searching workflow described in Section 2, and use the real-world data 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

Fig. 13 shows the structure of our simulation environment. SwinCloud is a cloud computing simulation environment built on the computing facilities at Swinburne University of Technology which takes advantage of the existing SwinGrid system [35]. We install VMWare software [33] on SwinCloud, so that it can offer unified computing and storage resources. By utilising the unified resources, we set up data centres that can host applications. In every data centre, Hadoop [19] is installed, which can facilitate the MapReduce computing paradigm and distributed data management. SwinDeW-C (Swinburne Decentralised Workflow for Cloud) [23] is a cloud workflow system developed based on SwinDeW [34] and SwinDeW-G [35]. It runs on SwinCloud, and can interpret and execute workflows, send and retrieve, and save and delete datasets 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 simulations, we facilitate our strategy in SwinDeW-C to manage the intermediate datasets in the simulation cloud.

To evaluate the performance of our strategy, we run six simulation strategies together and compare the total costs 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 intermediate datasets that have high generation cost; (4) store the intermediate datasets that are most often used; (5) the dependency-based strategy reported in [36,38], in which we store the datasets by comparing their generation cost rates and storage cost rates; and (6) the minimum cost storage strategy found by the CTT-SP algorithm in our benchmarking.

We have run a large number of simulations with different parameters to evaluate the performance of our benchmark. We evaluate some representative results in this section.

5.2. General (random) workflow simulations

To evaluate the overall performance of our strategy in a general manner, we have run a large number of random simulations with the six strategies introduced earlier. In general simulations, we use randomly generated workflows to construct the IDG, and give every intermediate dataset a random size, generation time, and usage rate, and then run the workflows under different pricing models. We compare the total system costs over 30 days for different strategies, which show the cost effectiveness of the strategies in comparison to our minimum cost benchmark.

We pick one test case as a representative. In this case, we let the workflow randomly generate 50 intermediate datasets, each with a random size ranging from 100 GB to 1 TB. The dataset generation time is also random, ranging from 1 to 10 h. The usage rate (time between every usage) is again randomly ranging from 1 day to 10 days. The prices of cloud services follow Amazon clouds' cost model, i.e. \$0.1 per CPU instance-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 the benchmark to evaluate the other five strategies introduced in Section 5.1. More random simulation cases can be found from the URL given in Section 5.1.

Fig. 14 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 percentages 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 Fig. 14, we can clearly see the cost effectiveness of different strategies compared with the benchmark, where storing the top 10% generation cost datasets turns out to be the most cost-effective strategy in this case. But the system cost is still much higher than the minimum cost benchmark.

Then we evaluate the storing often used datasets strategy by comparing with the benchmark. We still run simulations of strategies that store different percentages of datasets based on their usage rates. Fig. 15 shows the comparison of the total system costs over 30 days, where we can clearly see the cost effectiveness of different strategies compared with the benchmark. Also, the strategy of storing the top 10% often used datasets turns out to be the most cost-effective one in this case. Compared to Fig. 14, the strategy of storing often used datasets is more cost effective than storing high generation cost datasets, but it is again still much higher than the minimum cost benchmark.

The intermediate dataset storage strategies reported in [36,38] are also based on the IDG, which has considered the data dependencies in calculating the datasets' generation cost and storage cost. Fig. 16 shows the comparison of the dependency-based strategies with the minimum cost benchmark. In the dependency-based static strategy, datasets' storage statuses are decided when they are first generated in the system by comparing their generation cost rates and storage cost rates, and in the dependency-based dynamic strategy, whenever the datasets are regenerated in the system during the runtime, their storage statuses are recalculated and dynamically changed, and other datasets' storage statuses may also be adjusted accordingly. In Fig. 16, we can see that

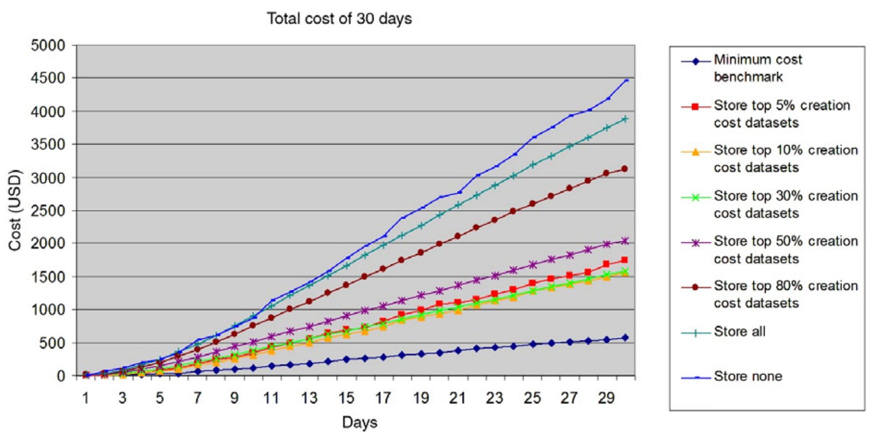


Fig. 14. Cost effectiveness evaluation of the "store high generation cost datasets" strategy.

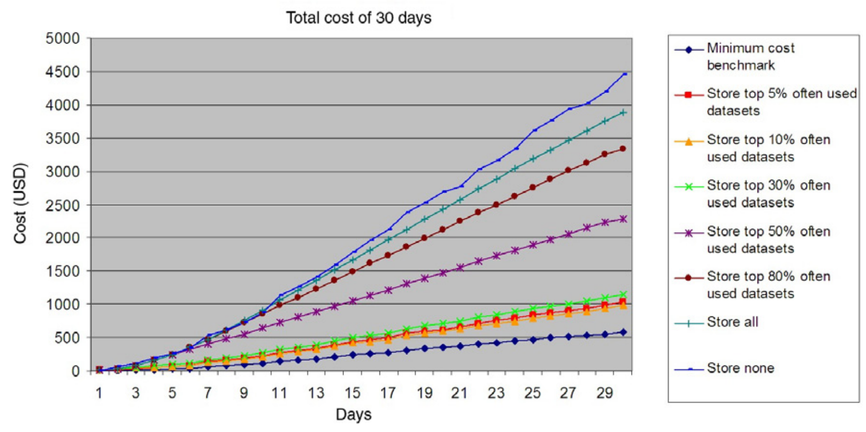


Fig. 15. Cost effectiveness evaluation of the "store often used datasets" strategy.

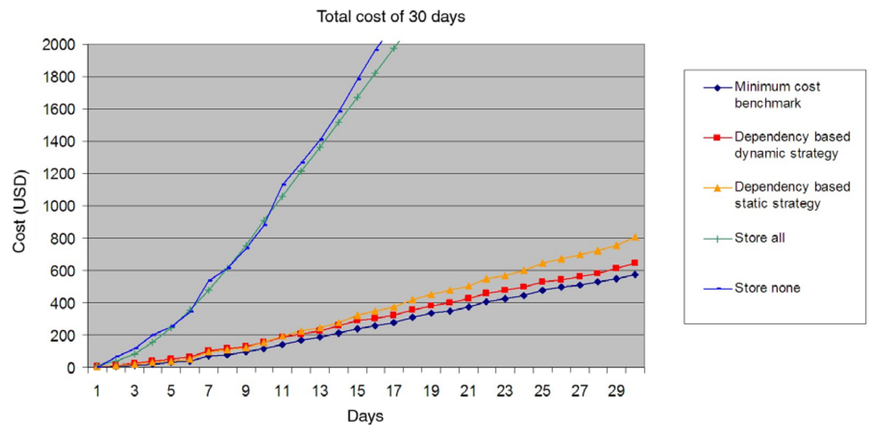


Fig. 16. Cost-effectiveness evaluation of the dependency-based strategy.

the dependency-based strategies have a good performance, which are more cost effective than the strategies depicted in Figs. 14 and 15. In particular, for the dynamic strategy, based on the adjustment of the datasets' storage statuses in the runtime of the system, its cost is close to the minimum cost benchmark that is calculated in the build time.

5.3. *Specific pulsar searching workflow simulations*

The general (random) workflow simulations demonstrate how to utilise our minimum cost benchmark to evaluate the cost effectiveness of different intermediate dataset storage strategies. Next we utilise it for the specific pulsar searching workflow introduced in Section 2 and show how the benchmark works in a real-world 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 Fig. 17, as well as the sizes and generation times of these intermediate datasets. The generation times are from running this workflow on Swinburne Supercomputer [30], and for simulations, we assume that, in the cloud system, the generation times of these intermediate datasets are the same.

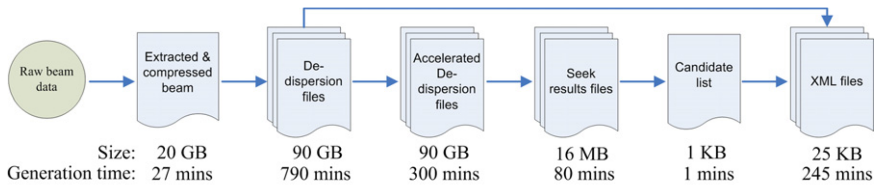

Raw beam data
Extracted & compressed beam
De-dispersion files
Accelerated De-dispersion files
Seek results files
Candidate list
XML files
Size: 20 GB 90 GB 90 GB 16 MB 1 KB 25 KB
Generation time: 27 mins 790 mins 300 mins 80 mins 1 mins 245 mins

Fig. 17. IDG of the pulsar searching workflow.

Furthermore, we also assume that the prices of the cloud services follow Amazon clouds' cost model.

We have run the simulations based on the usage rates of intermediate datasets. From the 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 for pulsar candidates. Based on the scenario, we set the "de-dispersion files" to be used once every 2 days, and the rest of the intermediate datasets to be used once every 5 days. With 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, as shown in Fig. 18.

From Fig. 18, we can see that (1) the cost increase of the "store all" strategy is in 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 increase of the "store none" strategy is in a fluctuated line, because in this strategy all the costs are computation costs 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 increases of the "store high generation cost datasets" and "store often used datasets" strategies are in the middle band, and are much lower than the costs of the "store all" and "store none" strategies. The cost lines are only a little fluctuated, because the intermediate datasets are partially stored; (5)–(6) the dependency-based strategy has a good performance in this pulsar searching workflow, which is very close to the minimum cost benchmark.

Table 1 shows how the six strategies store the intermediate datasets in detail.

Since the intermediate datasets of this pulsar searching workflow are not very complicated, we can do some intuitive analyses on how to store them. For the accelerated de-dispersion files, although their generation cost is quite high, compared to the huge size, it is not worth storing them in the cloud. However, in the "store high generation cost datasets" strategy, the accelerated de-dispersion files are chosen to be stored. The final XML files are not used very often, but given the high generation cost and small size, they should be stored. However, in the "store often used datasets" strategy, these files are not chosen to be stored. For the de-dispersion files, by comparing their own generation cost rate and storage cost rate, the dependency-based strategy does not store them at the beginning, but stores them after they are used in the regeneration of other datasets.

6. Related work

Compared to distributed computing systems like clusters and grids, a cloud computing system has a cost benefit [4]. Assuncao 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 [22] 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 [31] introduces a scientific cloud architecture for a data centre, and the Nimbus [21] toolkit, which can directly turn a cluster into a cloud, has already been used to build a cloud for scientific applications. In terms of the cost benefit, the work by Deelman et al. [16] 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, and shows that applications running on the cloud have cost benefits. However, our work studies how to reduce the cost if we run scientific workflows on the cloud. In [16], 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 [1], Adams et al. propose a model to represent the trade-off of computation cost and storage cost, but they have not given the strategy to find this trade-off. In [36,38], Yuan et al. propose a cost-effective strategy for intermediate data storage in scientific cloud workflows systems that takes data dependency into consideration. Comparison with the benchmark proposed in this paper indicates that the strategy in [36,38] has a good performance, but does not achieve the minimum cost of the system. In this paper, the minimum cost benchmarking contains an innovative algorithm (i.e. the CTT-SP algorithm) that can find the minimum cost strategy on demand for storing intermediate datasets in scientific cloud workflow systems based on the historic usage information of the datasets.

The study of data provenance is important for our work. Due to the importance of data provenance in scientific applications, much research about recording data provenance of the system has been done [18,9]. Some of this work is especially for scientific workflow systems [9]. Some popular scientific workflow systems, such as Kepler [24], have their own system to record provenance during the workflow execution [2]. In [28], 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, normally it is not practical to run scientific applications among different cloud service providers, for the following reasons.

(1) The datasets in scientific applications 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 common practice to transfer application data, and it is also considered to be the most

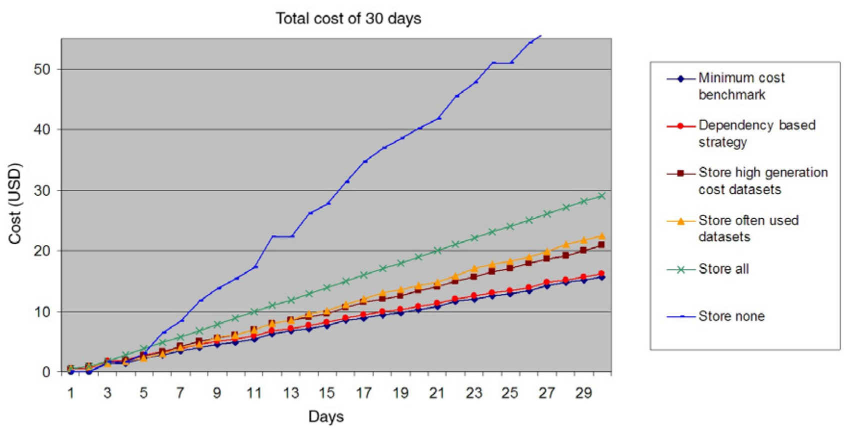


Fig. 18. Evaluation of strategies in storing the pulsar seeking workflow data.

Table 1
Pulsar searching workflow's intermediate dataset storage status in the six strategies.

Strategies	Datasets					
	Extracted beam	De-dispersion files	Accelerated de-dispersion files	Seek results	Pulsar candidates	XML files
(1) Store all	Stored	Stored	Stored	Stored	Stored	Stored
(2) Store none	Deleted	Deleted	Deleted	Deleted	Deleted	Deleted
(3) Store high generation cost datasets	Deleted	Stored	Stored	Deleted	Deleted	Stored
(4) Store often used datasets	Deleted	Stored	Deleted	Deleted	Deleted	Deleted
(5) Dependency-based strategy	Deleted	Stored (deleted initially)	Deleted	Stored	Deleted	Stored
(6) Minimum cost benchmark	Deleted	Stored	Deleted	Stored	Deleted	Stored

efficient way to transfer terabytes of data [4]. Nowadays, express delivery companies can deliver hard disks nationwide by the end of the next day and worldwide in 2 or 3 days. In contrast, transferring one terabyte of data via the Internet would take more than 10 days at a speed of 1 MB/s. To break the bandwidth limitation, some institutions have set up 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 would still prefer to ship hard disks. Furthermore, building fibre connections is expensive, and they are not yet widely used in the Internet. Hence, transferring scientific application data between different cloud service providers via the Internet is not efficient.

(2) Cloud service providers place high cost on data transfer into and out of their data centres. In contrast, data transfers 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 may not be cost effective. The cloud service providers charge a high price for 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 did a promotion that placed a zero price on data transferred into its data centres, until June 30, 2010, which means the users could 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 reasons discussed above, the most efficient and cost-effective way to run scientific applications in a cloud is to keep all the data and run the applications with one cloud service provider; a similar conclusion is also stated in [16]. Hence, in the strategy stated in this paper, we did not take the data transfer cost into consideration. However, some scientific applications may have to run in a distributed manner [13,12], because the required datasets are distributed, some with fixed locations. In these cases, data transfer is inevitable, and a data placement strategy [37] would be needed to reduce the data transfer cost.

8. Conclusions and future work

In this paper, based on an astrophysics pulsar searching workflow, we have examined the unique features of intermediate dataset storage in scientific cloud workflow systems and developed a novel algorithm that can find the minimum cost intermediate dataset storage strategy on demand. This strategy achieves the best trade-off of computation cost and storage cost of the cloud resources, which can be utilised as the minimum cost benchmark for evaluating the cost effectiveness of other dataset storage strategies. Simulation results of both general (random) workflows and the specific pulsar searching workflow demonstrate that our benchmarking serves well for such a purpose.

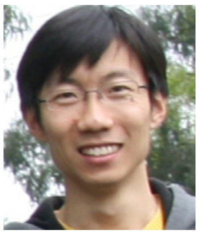
Our current work is based on Amazon clouds' cost model and assumes that all the application data are stored with a single cloud service provider. However, sometimes scientific workflows have to run in a distributed manner since some application data are distributed and may have fixed locations. In these cases, data transfer is inevitable. In the future, we will further develop some data placement strategies in order to reduce data transfer among data centres. Furthermore, to widely utilise our benchmarking, models of forecasting intermediate dataset usage rates can be further studied. Such a model must be flexible in order to be adapted to different scientific applications. Due to the dynamic nature of cloud computing environments, the minimum cost benchmarking of scientific cloud workflows needs to be enhanced, where the minimum cost benchmark should be able to dynamically adjust according to the change of datasets usages at runtime.

Acknowledgments

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 from Mr. B. Gibson.

References

[1] I. Adams, D.D.E. Long, E.L. Miller, S. Pasupathy, M.W. Storer, Maximizing efficiency by trading storage for computation, in: Workshop on Hot Topics in Cloud Computing, HotCloud'09, San Diego, CA, 2009, pp. 1–5.
[2] I. Altintas, O. Barney, E. Jaeger-Frank, Provenance collection support in the Kepler scientific workflow system, in: International Provenance and Annotation Workshop, Chicago, Illinois, USA, 2006, pp. 118–132.
[3] Amazon Cloud Services. http://aws.amazon.com/ (accessed on 12.08.10).
[4] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, Above the clouds: a Berkeley view of cloud computing, Technical Report UCB/EECS-2009-28, University of California at Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf (accessed on 12.08.10).
[5] M.D.d. Assuncao, A.d. Costanzo, 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, HPDC'09, Garching, Germany, 2009, pp. 1–10.
[6] ATNF Parkes Swinburne Recorder.http://astronomy.swin.edu.au/pulsar/?topic=apsr (accessed on 12.08.10).
[7] Australia Telescope National Facility. http://www.atnf.csiro.au/ (accessed on 12.08.10).
[8] Australia Telescope, Parkes Observatory. http://www.parkes.atnf.csiro.au/ (accessed on 12.08.10).
[9] Z. Bao, S. Cohen-Boulakia, S.B. Davidson, A. Eyal, S. Khanna, Differencing provenance in scientific workflows, in: 25th IEEE International Conference on Data Engineering, ICDE'09, Shanghai, China, 2009, pp. 808–819.
[10] R. Bose, J. Frew, Lineage retrieval for scientific data processing: a survey, ACM Computing Surveys 37 (2005) 1–28.
[11] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (2009) 599–616.
[12] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, Giggle: a framework for constructing scalable replica location services, in: ACM/IEEE Conference on Supercomputing, SC'02, Baltimore, Maryland, 2002, pp. 1–17.
[13] A. Chervenak, E. Deelman, M. Livny, M.-H. Su, R. Schuler, S. Bharathi, G. Mehta, K. Vahi, Data placement for scientific applications in distributed environments, in: 8th Grid Computing Conference, Austin, Texas, USA, 2007, pp. 267–274.
[14] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, M. Livny, Pegasus: mapping scientific workflows onto the grid, in: European Across Grids Conference, Nicosia, Cyprus, 2004, pp. 11–20.
[15] E. Deelman, A. Chervenak, Data management challenges of data-intensive scientific workflows, in: IEEE International Symposium on Cluster Computing and the Grid, CCGrid'08, Lyon, France, 2008, pp. 687–692.
[16] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: the montage example, in: ACM/IEEE Conference on Supercomputing, SC'08, Austin, Texas, 2008, pp. 1–12.
[17] I. Foster, Z. Yong, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Grid Computing Environments Workshop, GCE'08, Austin, Texas, USA, 2008, pp. 1–10.
[18] P. Groth, L. Moreau, Recording process documentation for provenance, IEEE Transactions on Parallel and Distributed Systems 20 (2009) 1246–1259.
[19] Hadoop. http://hadoop.apache.org/ (accessed on 12.08.10).
[20] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, J. Good, On the use of cloud computing for scientific workflows, in: 4th IEEE International Conference on e-Science, Indianapolis, Indiana, USA, 2008, pp. 640–645.
[21] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, M. Tsugawa, Science clouds: early experiences in cloud computing for scientific applications, in: First Workshop on Cloud Computing and its Applications, CCA'08, Chicago, Illinois, USA, 2008, pp. 1–6.
[22] D. Kondo, B. Javadi, P. Malecot, F. Cappello, D.P. Anderson, Cost-benefit analysis of cloud computing versus desktop grids, in: IEEE International Symposium on Parallel & Distributed Processing, IPDPS'09, Rome, Italy, 2009, pp. 1–12.
[23] X. Liu, D. Yuan, G. Zhang, J. Chen, Y. Yang, SwinDeW-C: a peer-to-peer based cloud workflow system for managing instance intensive applications, in: Handbook of Cloud Computing, Springer, 2010, pp. 309–332. http://www.ict.swin.edu.au/personal/xliu/papers/SwinDeW-C.pdf.
[24] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, Scientific workflow management and the Kepler system, Concurrency and Computation: Practice and Experience (2005) 1039–1065.
[25] C. Moretti, J. Bulosan, D. Thain, P.J. Flynn, All-Pairs: an abstraction for data-intensive cloud computing, in: IEEE International Parallel & Distributed Processing Symposium, IPDPS'08, Miami, Florida, USA, 2008, pp. 1–11.
[26] P. Odifreddi, Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers, vol. 125, Elsevier, 1992, pp. ii-xi, 1–668.
[27] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, P. Li, Taverna: a tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20 (2004) 3045–3054.
[28] L.J. Osterweil, L.A. Clarke, A.M. Ellison, R. Podorozhny, A. Wise, E. Boose, 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, 2008, pp. 319–329.
[29] Y.L. Simmhan, B. Plale, D. Gannon, A survey of data provenance in e-science, SIGMOD Record 34 (2005) 31–36.
[30] Swinburne high performance supercomputing facility. http://astronomy.swin.edu.au/supercomputing/ (accessed on 12.08.10).
[31] L. Wang, J. Tao, M. Kunze, A.C. Castellanos, D. Kramer, W. Karl, Scientific cloud computing: early definition and experience, in: 10th IEEE International Conference on High Performance Computing and Communications, HPCC'08, Dalin, China, 2008, pp. 825–830.
[32] A. Weiss, Computing in the cloud, ACM Networker 11 (2007) 18–25.
[33] VMware. http://www.vmware.com/ (accessed on 12.08.10).
[34] J. Yan, Y. Yang, G.K. Raikundalia, SwinDeW—a P2P-based decentralized workflow management system, IEEE Transactions on Systems, Man and Cybernetics, Part A 36 (2006) 922–935.
[35] Y. Yang, K. Liu, J. Chen, J. Lignier, H. Jin, Peer-to-Peer based grid workflow runtime environment of SwinDeW-G, in: IEEE International Conference on e-Science and Grid Computing, Bangalore, India, 2007, pp. 51–58.
[36] D. Yuan, Y. Yang, X. Liu, J. Chen, A cost-effective strategy for intermediate data storage in scientific cloud workflows, in: 24th IEEE International Parallel & Distributed Processing Symposium, IPDPS'10, Atlanta, Georgia, USA, 2010, pp. 1–12.
[37] D. Yuan, Y. Yang, X. Liu, J. Chen, A data placement strategy in scientific cloud workflows, Future Generation Computer Systems 26 (2010) 1200–1214.
[38] D. Yuan, Y. Yang, X. Liu, G. Zhang, J. Chen, A data dependency based strategy for intermediate data storage in scientific cloud workflow systems, Concurrency and Computation: Practice and Experience (2010) (http://dx.doi.org/10.1002/cpe.1636).



Dong Yuan was born in Jinan, China. He received his B.Eng. degree in 2005 and M.Eng. degree in 2008, both from Shandong University, Jinan, China, in computer science.

He is currently a Ph.D. student in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Victoria, Australia. His research interests include data management in workflow systems, scheduling and resource management, grid and cloud computing.

Yun Yang was born in Shanghai, China. He received his B.S. degree from Anhui University, Hefei, China, in 1984, his M.Eng. degree from the University of Science and Technology of China, Hefei, China, in 1987, and his Ph.D. degree from the University of Queensland, Brisbane, Australia, in 1992, all in computer science.

He is currently a Full Professor in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Victoria, Australia. Prior to joining Swinburne as an Associate Professor, he was a Lecturer and Senior Lecturer at Deakin University during th period 1996–1999. Before that, he was a (Senior) Research Scientist at DSTC Cooperative Research Centre for Distributed Systems Technology during the period 1993–1996. He also worked at the Beijing University of Aeronautics and Astronautics during the period 1987–1988. He has co-edited two books and published more than 170 papers in journals and refereed conference proceedings. His current research interests include software technologies, p2p/grid/cloud workflow systems, service-oriented computing, cloud computing, and e-learning.

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

Jinjun Chen received his Ph.D. degree from Swinburne University of Technology, Melbourne, Victoria, Australia. His thesis was granted a Research Thesis Excellence Award. He received the Swinburne Vice Chancellor's research award in 2008. He is a core executive member of the IEEE Technical Committee of Scalable Computing and the coordinator of the IEEE TCSC technical area of Workflow Management in Scalable Computing Environments. He is the Editor-in-Chief of the Springer book series on Advances in Business Process and Workflow Management (http://www.swinflow.org/books/springer/SpringerBook.htm) and Editor-in-Chief of the Nova book series on Process and Workflow Management and Applications (http://www.swinflow.org/books/nova/NovaBook.htm). He has guest edited or is editing several special issues in quality journals such as in IEEE Transactions on Automation Science and Engineering. He has been involved in the organization of many conferences, and was awarded the IEEE Computer Society Service Award (2007).

He has published more than 50 papers in journals and conference proceedings such as those of ICSE2008 and ACM TAAS. His research interests include scientific workflow management and applications, workflow management and applications in Web service or SOC environments, workflow management and applications in grid (service)/cloud computing environments, software verification and validation in workflow systems, QoS and resource scheduling in distributed computing systems such as cloud computing, and service-oriented computing (SLA and composition).