An Algorithm for Finding the Minimum Cost of Storing and Regenerating Datasets in Multiple Clouds

Dong Yuan, *Member, IEEE,* Lizhen Cui, Wenhao Li, Xiao Liu, *Member, IEEE* and Yun Yang, *Senior Member, IEEE*

Abstract—The proliferation of cloud computing allows users to flexibly store, re-compute or transfer large generated datasets with multiple cloud service providers. However, due to the pay-as-you-go model, the total cost of using cloud services depends on the consumption of storage, computation and bandwidth resources which are three key factors for the cost of IaaS-based cloud resources. In order to reduce the total cost for data, given cloud service providers with different pricing models on their resources, users can flexibly choose a cloud service to store a generated dataset, or delete it and choose a cloud service to regenerate it whenever reused. However, finding the minimum cost is a complicated yet unsolved problem. In this paper, we propose a novel algorithm that can calculate the minimum cost for storing and regenerating datasets in clouds, i.e. whether datasets should be stored or deleted, and furthermore where to store or to regenerate whenever they are reused. This minimum cost also achieves the best trade-off among computation, storage and bandwidth costs in multiple clouds. Comprehensive analysis and rigid theorems guarantee the theoretical soundness of the paper, and general (random) simulations conducted with popular cloud service providers' pricing models demonstrate the excellent performance of our approach.

Index Terms—Cloud Computing; Data Storage and Regeneration; Minimum Cost

1 Introduction

In recent years, cloud computing is emerging as the latest computing paradigm which provides redundant, inexpensive and scalable resources on demand to users [1] [2]. IaaS (Infrastructure as a Service) is a very popular way to deliver services in the cloud [3], where users can deploy their applications in unified cloud resources such as computing, storage and network services without any infrastructure investments. However, along with the convenience brought by using on-demand cloud services, users have to pay for the resources used according to the pay-as-you-go model, which can be substantial. Especially, nowadays applications are getting more and more data intensive [4], where the generated data are often gigabytes, terabytes, or even petabytes in size. These generated data contain important intermediate or final results of computation, which may need to be stored for reuse [5]. Hence, cutting the cost of cloud-based data management in a pay-as-you-go fashion becomes a big concern for deploying applications in cloud computing environment.

Cloud computing has such a fast growing market, more and more cloud service providers appear with different prices of computation, storage and bandwidth resources [6] [7]. As unlimited storage and processing power can be easily obtained on-demand from different commercial service providers like utilities, users have multiple options to cope with the large generated application data, e.g., datasets d_1, d_2 … d_8 in Figure 1. Specifically, users can store all data in the cloud and simply pay for the storage cost, and alternatively, they can delete some data to save the storage cost and pay for the computation cost to regenerate them whenever they are reused, e.g, datasets d_2, d_6 and d_8 are deleted in Figure 1. Furthermore, users can also change to cheaper service providers to store or to regenerate data with paying for the bandwidth cost for data transfer[1]. Hence, there is a trade-off among computation, storage and bandwidth in clouds, where different storage and regeneration strategies lead to different total costs for storing the generated application data. In light of this, users need comprehensive understanding of cost in clouds in order to take advantage of the cost-effectiveness of cloud computing, especially for storing and regenerating data with multiple cloud service providers[2].

Finding the trade-off among computation, storage and bandwidth costs in clouds is a complicated problem. Different cloud service providers have different prices on their re-

- *D. Yuan is with the University of Sydney, Sydney, NSW 2006, Australia. E-mail: dong.yuan@sydney.edu.au.*
- *L. Cui and W. Li are with the Shandong University, Jinan, Shandong 250101, China. E-mail: clz@sdu.edu.cn ayumi_5420467@hotmail.com*
- *X. Liu is with the Software Engineering Institute, East China Normal University, Shanghai 200062, China. E-mail: xliu@sei.ecnu.edu.cn.*
- *Y. Yang is with the Anhui University, Hefei, Anhui 230039, China and Swinburne University of Technology, Hawthorn, Melbourne, VIC 3122, Australia. E-mail: yyang@swin.edu.au.*

*Please note that all acknowledgments should be placed at the end of the paper, before the bibliography (**note that corresponding authorship is not noted in affiliation box, but in acknowledgment section**).*

[1] The "vendor lock-in" issue may bring extra cost in regenerating application data with different cloud service providers. In this paper, we do not consider this extra cost in clouds, and we have listed this issue as our future work.

[2] In this paper, term "cloud service provider(s)" refers to IaaS provider, and term "user(s)" refers to users of IaaS providers who are often Software as a Service (SaaS) and Platform as a Service (PaaS) providers.

xxxx-xxxx/0x/$x.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

sources and datasets also have different sizes, generation times and usage frequencies. Intuitively, some heuristics can be applied. For example, we can store the frequently used data which have high generation costs in cloud services with cheaper storage resources. Also, we can delete the less frequently used data which have large sizes but small regeneration costs, and regenerate them in cloud services with cheaper computation resources. Not only those, but data also have dependencies in clouds, i.e. the complex generation relationships. The data regeneration cost depends on their stored provenance data; hence the change of storage status of any data will impact regeneration cost of the data derived from them.

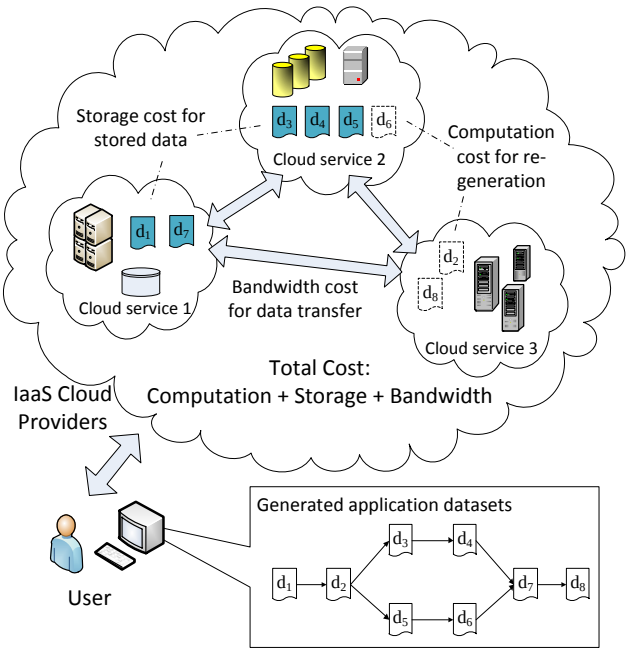


Figure 1. Storage and regeneration of application datasets in clouds

In this paper, we propose a novel GT-CSB algorithm that can find the best trade-off among computation, storage and bandwidth costs in clouds. This trade-off is represented by the theoretical minimum cost strategy for storing and regenerating application data among multiple cloud service providers. This minimum cost is a very important reference for cloud users in the following three aspects: 1) it can be used to design minimum cost benchmarking approaches for evaluating the cost effectiveness in clouds; 2) it can guide cloud users to develop cost effective storage strategies for their applications; and 3) it can demonstrate the constitution of different costs in clouds and help users to understand the impact of different workloads on the total cost.

The remainder of this paper is organised as follows. Section 2 discusses the related work. Section 3 introduces a motivating example of our research and presents the preliminaries including some important concepts and the trade-off based cost model in clouds. Section 4 presents our GT-CSB algorithm to find the best trade-off among computation, storage and bandwidth in clouds. Section 5 discusses the utilisation of the GT-CSB algorithm in designing cost effective storage strategy and minimum cost benchmarking approaches. Section 6 describes our experimental results for evaluation. Section 7 summarises our conclusions and points out future work.

2 Related Work

In the research area of resource management, much work has been done about resource negotiation [8], replica placement [9] and multi-tenancy in clouds [10], by taking advantage of different types of resources. Our work also investigates different types of resources, but we mainly focus on the trade-offs among them. Another important foundation for our work is the research on data provenance [11]. More specifically, Osterweil et al. [12] present how to generate a provenance based data derivation graph for execution of a workflow. Foster et al. [13] propose the concept of virtual data in the Chimera system, which enables the automatic regeneration of data when needed. Recently, research on data provenance in cloud computing systems has also appeared [14]. Based on the above research, in this paper, we utilise a Data Dependency Graph (DDG) which is based on data provenance. DDG depicts the generation relationships of all the generated data in clouds, with which we can manage where the data are stored or how to regenerate them. Since graph theory is a useful tool in computer science [15] [16], we create a transitive graph based on DDG and propose an algorithm that can find the best trade-off among computation, storage and bandwidth in clouds.

Plenty of research has been done with regard to the trade-off between computation and storage. The Nectar system [17] is designed for automatic management of data and computation in data centres, where obsolete data are deleted and regenerated whenever reused in order to improve resource utilisation. In [18], 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 [19], Adams et al. propose a model to represent the trade-off of computation cost and storage cost. In [20], the authors propose the CTT-SP algorithm that can find the best trade-off between computation and storage in the cloud, based on which a highly cost-effective and practical strategy is developed for storing datasets with one cloud service provider [21]. However, the above work did not consider the bandwidth cost into the trade-off model.

As the trade-off among different costs is an important issue in the cloud, some research has already embarked on this issue to a certain extent. In [22], Joe-Wong et al. investigate computation, storage and bandwidth resources allocation in order to achieve a trade-off between fairness and efficiency. In [23], Baliga et al. investigate the trade-off among computation, storage and bandwidth at the infrastructure level of cloud

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

systems, where reducing energy consumption is the main research goal. In [24], Chen et al. further analyse the impacts of computation, storage and bandwidth on the total energy consumption by investigating different types of applications in the cloud. In [6], Agarwala et al. transform application data to certain formats and store them with different cloud services in order to reduce storage cost in the cloud, but data dependency and the option of data regeneration are not considered in their work. In [25], Wu et al. propose *SPANStore* which can cost-effectively place data replicas among geo-distributed data centres to reduce the access delay. It only finds the trade-off between storage and bandwidth. In our prior work [26], we propose the T-CSB algorithm which can find a trade-off among Computation, Storage and Bandwidth costs (T-CSB). However, it is not generic because there is a strong assumption that all the computation (i.e. data regeneration) must be conducted in one cloud service, which poses certain limitation.

In this paper, we propose the GT-CSB algorithm, which can find a *G*eneric best *T*rade-off among *C*omputation, *S*torage and *B*andwidth (GT-CSB) in clouds. This generic best trade-off represents the minimum cost storage and regeneration strategy, in which data can be stored or regenerated with any cloud service providers.

3 Motivating Example and Preliminaries

In this section, we first present a motivating example, in which our GT-CSB algorithm can be used. Then we introduce some preliminaries, including a classification of application data in clouds, the concept of Data Dependency Graph (DDG) and the trade-off based cost model among computation, storage and bandwidth costs for datasets storage and regeneration in cloud computing.

3.1 Motivating Example

In order to explore the usage scenarios of our approach, we have investigated some data intensive applications, which are 1) a Finite Element Modelling (FEM) application in Structural Mechanics, 2) a Climatological Analyses Application in Meteorology and 3) a Pulsar Searching Application in Astrophysics. Due to the page limit, we only present the FEM application in this section as a motivating example and provide the other two applications in Supplementary Materials.

Finite Element Modelling (FEM) is an important and widely used method for impact test of objects, where classic applications are Hopkinson pressure bar test, gas gun impact test, drop hammer test, etc. At Swinburne University of Technology, researchers of the Structural Mechanics Research Group conduct FEM simulations of Aluminium Honeycombs under dynamic out-of-plane compression to analyse the impact behaviour of the material and structure [27]. In their research, numerical simulations of the dynamic out-of-plane compression are conducted with ANSYS/LS-DYNA software, which is a powerful FEM tool for modelling non-linear mechanics of solids, fluids, gases and their interaction. The FEM application has four major steps as shown in Figure 2.

From Figure 2, at beginning, based on the researchers' design, the object with special structure (i.e. the honeycombs structure in this example) for FEM analysis is generated in the Object Modelling step. Then, researchers specify more detailed parameters of the object model in the FEM Initiation step, e.g. material of the object and elements for modelling. Based on the well-defined model, researchers can run different FEM simulations according to requirements of the experiment, e.g. speed of the compression and time interval for recording data. This is the most time consuming and important step in the FEM application, which also generates the largest volume of data as simulation results. Depending on the speed of the compression, the computation time of this step varies from several hours to around one hundred hours, while depending on the time interval for recording data, the size of generated data varies from gigabytes to hundreds of gigabytes. These data are very important for researchers, based on which the simulation results can be demonstrated in various ways for analysis.

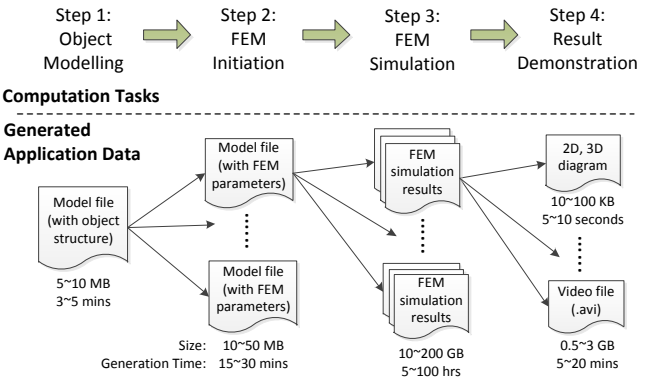


Figure 2. Overview of FEM application

With cloud computing, researchers can easily obtain computting resources to run the application, e.g., Amazon EC2[3]. As time goes on, large volumes of simulation results are accumulated (e.g., Step 3 in Figure 2). As these data have high generation cost, they should be stored for reuse. The often-used datasets can be stored in Amazon S3[4], and for the rarely used datasets, it is more cost effective to transfer them to Amazon Glacier[5] for storage. Futhermore, for some rarely used datasets that do not have high generation cost (e.g., the vedio file in Step 4 in Figure 2), it is more cost effective to delete them and regenerate whenever reused. In this paper, the proposed GT-CSB algorithm can automatically calculate the minimum cost storage and regeneration strategy for the application data in multiple clouds.

[3] http://aws.amazon.com/ec2/
[4] http://aws.amazon.com/s3/
[5] http://aws.amazon.com/glacier/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

3.2 Preliminaries

3.2.1 Classification of Application Data in Clouds

In general, there are two types of data stored in clouds, original data and generated data.

1) *Original data* are the data uploaded by users, for example, in scientific applications they are usually the raw data collected from the devices in the experiments. For these data, users need to decide whether they should be stored or deleted since they cannot be regenerated by the system once deleted. As cost of storing *original data* is fixed, they are **not** considered in the scope of this paper.

2) *Generated data* are the data newly produced in the cloud while the applications are running. They are the intermediate or final computation results of the applications, which can be reused in the future. For these data, their storage can be decided by the system since they can be regenerated if their provenance is known. Hence, our storage and regeneration strategy is **only** applied to the generated *data* in the cloud that can automatically decide the storage status of generated datasets in applications. In this paper, we refer *generated data* as **dataset(s)**.

3.2.2 Data Dependency Graph (DDG)

In this paper, we use DDG to represent generated datasets and their relationships. DDG [20] is a directed acyclic graph (DAG) which is based on data provenance in applications. All the datasets once generated in the cloud, whether stored or deleted, their references are recorded in DDG. In other words, it depicts the generation relationships of datasets, with which the deleted datasets can be regenerated from their nearest existing preceding datasets. Figure 3 depicts a simple DDG, where every node in the graph denotes a dataset, and the whole DDG will be the input of the GT-CSB algorithm. We denote dataset d_i in DDG as $d_i \in DDG$. Furthermore, d_1 pointing to d_2 means that d_1 is used to generate d_2; 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; d_4 and d_6 pointing to d_7 means that d_4 and d_6 are used together to generate d_7.

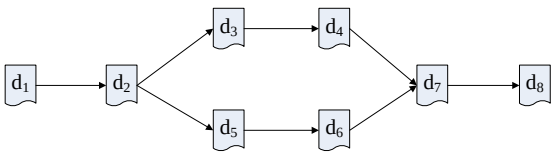


Figure 3. A simple Data Dependency Graph (DDG)

To better describe the relationships of datasets in DDG, we define a symbol: $\rightarrow$, which 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 DDG. For example, in Figure 3's DDG, 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, i.e.

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

3.2.3 Datasets Storage and Regeneration Cost Model

In a commercial cloud computing environment, service providers have their pricing models to charge users. In general, there are three basic types of resources in the cloud: computation, storage and bandwidth. In this paper, we facilitate our cost model in the cloud as follows:

$$Cost = Computation + Storage + Bandwidth$$

where the total cost of the datasets storage, *Cost*, is the sum of *Computation*, which is the total cost of computation resources used to regenerate datasets, *Storage*, which is the total cost of storage resources used to store the datasets, and *Bandwidth*, which is the total cost of bandwidth resources used for transferring datasets via the network.

In order to utilise the datasets storage and regeneration cost model, we present the following assumptions, denotations and definitions.

Assumptions: We assume that the application be deployed with m Cloud Service Providers, denoted as $CSP=\{c_1, c_2, \ldots c_m\}$, and each c_i has computation, storage and network services with different prices[6]. Furthermore, we assume there be n datasets in the DDG, denoted as $DDG=\{d_1, d_2, \ldots d_n\}$. For every dataset $d_i \in DDG$, it can be either stored with one of the cloud service providers or be deleted. If it is deleted, it can be regenerated with any one of the cloud service providers $c_j \in CSP$ whenever it needs to be reused.

Denotations: We use ***X, Y, Z*** to denote the computation cost, storage cost and bandwidth cost of datasets respectively. Specifically, for a dataset $d_i \in DDG$:

$X_{d_i}^{c_j}$ denotes the computation cost of regenerating dataset d_i from its direct predecessors in the DDG with cloud service provider c_j, which is calculated as the multiplication of the generation time of d_i and the price of computation resources of c_j;

$Y_{d_i}^{c_j}$ denotes the storage cost per time unit of storing dataset d_i with cloud service provider c_j, which is calculated as the multiplication of the size of d_i and the price of storage resources of c_j;

$Z_{d_i}^{c_k,c_j}$ denotes the bandwidth cost of transferring dataset d_i from cloud service provider c_k to c_j, which is calculated as the sum of outbound cost of d_i from c_k and inbound cost of d_i to c_j. Especially, if $c_k = c_j$ which means the

[6] Popular cloud services providers' cost models are based on these types of resources. For example, Amazon cloud services' prices are as follows:
\$0.10 per CPU instance hour for the computation resources;
\$0.15 per Gigabyte per month for the storage resources;
\$0.12 per Gigabyte bandwidth resources for data downloaded from Amazon via the Internet.
The prices may fluctuate from time to time according to market factors. As this paper's focus is cost-effectiveness, to simplify the problem, we assume that in one cloud service provider, the same types of computation resources are used for regenerating datasets, and the same types of storage resources are used for storing datasets.

data transfer is inside the same cloud service provider, then $Z_{d_i}^{c_k,c_j}=0$.

Furthermore, we use v_{d_i} to denote the usage frequency of d_i, which means how often d_i is reused. In cloud, datasets are often shared by many users on the Internet. Hence v_{d_i} cannot not be defined by a single user. In practice, it should be an estimated value from the dataset's usage history recorded in the system logs.

In a cloud computing environment, in order to regenerate a deleted dataset in the DDG, e.g. $d_i \in DDG$, first we need to find its stored provenance dataset(s), then we need to choose the cloud service providers to regenerate it.

Definition 1: Regeneration strategy of a deleted dataset is the selection of cloud service providers where the dataset is regenerated from its stored provenance dataset(s).

Hence, the regeneration cost of a dataset is twofold: 1) the bandwidth cost of transferring its stored provenance dataset(s) and intermediate dataset(s) to the corresponding cloud services, and 2) the computation cost of regenerating the dataset in these cloud services. However, different selections of cloud service providers for regeneration lead to different regeneration costs of the dataset.

We denote the minimum regeneration cost of dataset $d_i \in DDG$ as $minGenCost(d_i)$. In the next section, we will present the method of finding the minimum cost regeneration strategy in detail.

As this paper investigates the cost-effectiveness of long-term usage of cloud services, we introduce the concept of Cost Rate for datasets in clouds.

Definition 2: *Cost rate* of a dataset is the average cost spent on this dataset per time unit in clouds. For $d_i \in DDG$, we denote its cost rate as $CostR(d_i)$, which depends on the storage status of d_i. If d_i is stored, its cost rate is the storage cost per time unit in the corresponding cloud service. If d_i is deleted, its cost rate is the minimum regeneration cost multiplied by its usage frequency. Formally,

$$CostR(d_i)=\begin{cases} minGenCost(d_i)*v_{d_i}, & //\ d_i \ \ is \ \ deleted \\ Y_{d_i}^{c_j}, & //\ d_i \ \ is \ \ stored \ \ in \ \ c_j \end{cases}.$$

Based on the above definition, the Total Cost Rate of a DDG is the sum of cost rate of all the datasets in it, which is denoted as $TCR=\sum_{d_i \in DDG} CostR(d_i)$.

Definition 3: Storage strategy of a DDG is the storage status of all datasets in the DDG, i.e. whether the datasets are deleted or stored, and furthermore, which cloud services the datasets are stored in.

Furthermore, we can calculate the minimum total cost rate as a benchmark.

Definition 4: Minimum cost benchmark of a DDG is the minimum total cost rate for storing and regenerating datasets in the DDG with the minimum cost strategy, which is denoted as

$$TCR_{\min}=\min\left(\sum_{d_i \in DDG} CostR(d_i)\right).$$

Based on the definitions above, the minimum cost benchmark facilitates the minimum cost storage and regeneration strategy of datasets in a DDG, which is also the best trade-off among computation, storage and bandwidth costs in clouds. In the next section, we will present the design of our algorithm to derive this benchmark.

4 GT-CSB Algorithm

In the section, we present our novel GT-CSB algorithm. It can find the minimum cost storage and regeneration strategy for datasets, which represents the minimum cost benchmark in clouds. First, we briefly introduce the philosophy of the GT-CSB algorithm. Then, we present the detailed steps of the algorithm, as well as the proofs of the minimum cost benchmark in clouds. For the ease of illustration, we present the GT-CSB algorithm on linear DDG in this section. Linear DDG means a DDG with no branches, where each dataset in the DDG only has one direct predecessor and successor except the first and last datasets. The GT-CSB algorithm for general DDG is discussed in Section 5. At last, we analyse the computation complexity of the GT-CSB algorithm.

4.1 Overview of GT-CSB Algorithm

The GT-CSB algorithm is a graph based algorithm, which has the following main steps.

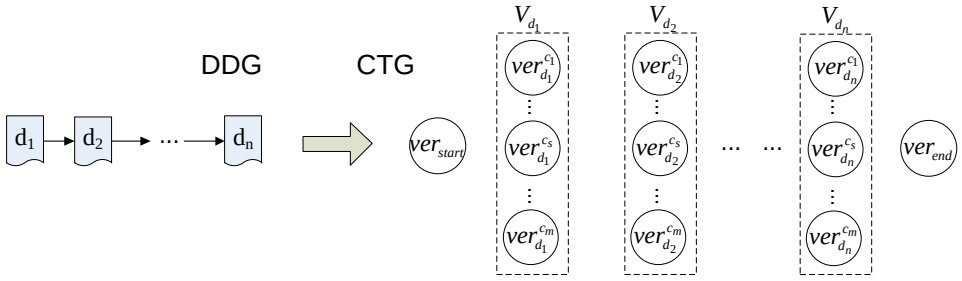
Step 1: Construct a Cost Transitive Graph (CTG) based on the DDG. First, for every dataset in the DDG, we create a set of vertices in the CTG representing services from different cloud service providers where datasets can be stored or regenerated. Then, we add edges to the CTG and guarantee that the paths in the CTG (from a start vertex to an end vertex) have one-to-one mapping to the storage strategies of datasets in the DDG.

Step 2: Set weights to the edges in the CTG. We present how to calculate the weights of edges that guarantee that the length of every path in the CTG (from a start vertex to an end vertex) equals to the total cost rate of the corresponding storage strategy with the minimum cost regeneration strategy of the deleted datasets.

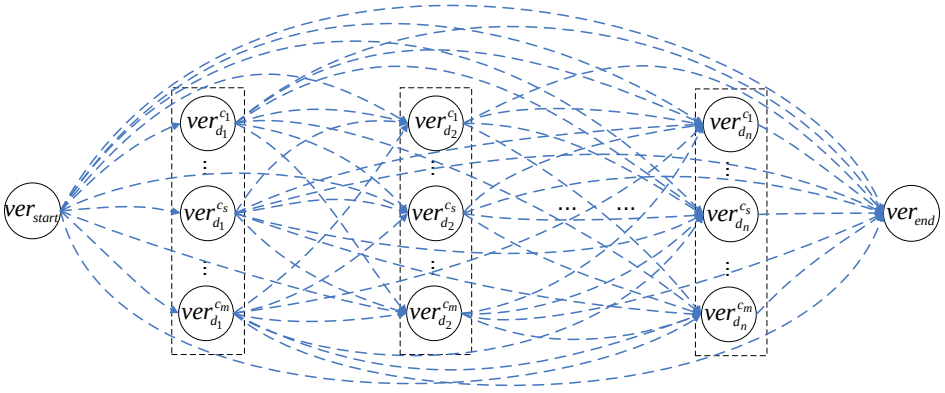
Step 3: Find the shortest path in the CTG. We can use the well-known Dijkstra shortest path algorithm (or Dijkstra algorithm for short) to find the shortest path in the CTG, which in fact represents both the minimum cost strategy for storing and regenerating datasets in the DDG with multiple cloud service providers, and the best trade-off among computation, storage and bandwidth costs in clouds.

Next, we will describe these steps in more detail on a linear DDG.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing



(a) Step 1.1: Create vertices for CTG



(b) Step 1.2: Create edges for CTG

Figure 4. Construct CTG from DDG

4.2 Detailed Steps in GT-CSB Algorithm on Linear DDG

Given a linear DDG with datasets {d_1, d_2 … d_n} and m cloud services {c_1, c_2 … c_m} for storage. The GT-CSB algorithm has the following steps:

Step 1: Construct CTG based on DDG.

Step 1.1: Create vertices for the CTG. As shown in Figure 4.(a), first, we create the start and end vertices, denoted as ver_{start} and ver_{end}. Then, for every $d_i \in DDG$, we create a vertex set $V_{d_i} = \left\{ver_{d_i}^{c_1}, ver_{d_i}^{c_2} ... ver_{d_i}^{c_m}\right\}$, where m is the number of cloud service providers with which d_i can be stored or regenerated. Hence $ver_{d_i}^{c_s}$ represents that dataset d_i is with cloud service provider c_s.

Step 1.2: Create directed edges for the CTG. As shown in Figure 4.(b), for every $ver_{d_i}^{c_s} \in CTG$, we add out-edges from it to all vertices which are created for the datasets succeeding to d_i in the DDG. Formally,

$$\left\{ver_{d_{i'}}^{c_{s'}} \middle| ver_{d_{i'}}^{c_{s'}} \in CTG \wedge (d_i, d_{i'} \in DDG \wedge d_i \rightarrow d_{i'})\right\}.$$

In other words, for any two vertices $ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}} \in CTG$ belonging to different datasets' vertex sets (i.e. $V_{d_i} \neq V_{d_{i'}}$), we create an edge between them. Formally,

$$ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}} \in CTG \wedge (d_i, d_{i'} \in DDG \wedge d_i \rightarrow d_{i'})$$
$$\Rightarrow \exists e < ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}} > .$$

Especially, we add out-edges from ver_{start} to all other vertices in the CTG, and we add in-edges from all other vertices in the CTG to ver_{end}.

Lemma 1: *The storage strategies for a DDG have one-to-one mapping to the paths from* ver_{start} *to* ver_{end} *in the CTG.*

Proof: For any path from ver_{start} to ver_{end}, we can find the vertices that it traverses. The storage strategy of the DDG for this path is to store the datasets with the cloud service providers that the traversed vertices represent, and delete the datasets that the path crosses over. On the contrary, given any storage strategy of the DDG, we can find the vertices in the CTG that represent the stored datasets and the corresponding cloud services providers. Then we can find the path from ver_{start} that traverses all the vertices found to ver_{end}. **Lemma 1 holds.**

Step 2: Set weights to the edges in the CTG.

Definition 5: For an edge $e < ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}} >$ in the CTG, we define its weight as the sum of Cost Rates of $d_{i'}$ and the datasets between d_i and $d_{i'}$, supposing that only d_i and $d_{i'}$ are stored with c_s and $c_{s'}$ and the datasets between d_i and $d_{i'}$ are all deleted. Formally,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

$$e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}> = CostR(d_{i'}) + \sum_{\{d_k | d_k \in DDG \wedge d_i \to d_k \to d_{i'}\}} CostR(d_k).$$

According to the definition of cost rate (Definition 2), we can get

$$e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}> = Y_{d_{i'}}^{c_{s'}} + \sum_{\{d_k | d_k \in DDG \wedge d_i \to d_k \to d_{i'}\}} \left(minGenCost(d_k) * v_{d_k} \right).$$

Based on the definition above, in order to calculate the weight of $e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}>$, we have to calculate the minimum regeneration cost of datasets between d_i and $d_{i'}$, i.e. $\left\{ d_k \middle| d_k \in DDG \wedge d_i \to d_k \to d_{i'} \right\}$.

In order to regenerate a dataset $d_j \in \left\{ d_k \middle| d_k \in DDG \wedge d_i \to d_k \to d_{i'} \right\}$, we need to start from the stored provenance dataset d_i, and regenerate all the datasets between d_i and d_j, i.e. $\left\{ d_k \middle| d_k \in DDG \wedge d_i \to d_k \to d_j \right\}$. All these can be regenerated with any cloud service providers in {c_1, c_2 … c_m}.

We design an iterative method to find all the minimum cost regeneration strategies for datasets in $\left\{ d_k \middle| d_k \in DDG \wedge d_i \to d_k \to d_{i'} \right\}$, which utilises the vertices in the CTG.

Definition 6: We define the value of a vertex $ver_{d_i}^{c_s} \in CTG$ as the minimum regeneration cost of d_i under the condition that regeneration of d_i from its direct predecessor is with cloud service provider c_s.

Based on this definition, for edge $e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}>$, the values of vertices are

$$\begin{cases} ver_{d_{i+1}}^{c_k} = Z_{d_i}^{c_s,c_k} + X_{d_{i+1}}^{c_k} \\ ver_{d_j}^{c_k} = \min_{h=1}^{m} \left\{ ver_{d_{j-1}}^{c_h} + Z_{d_{j-1}}^{c_h,c_k} \right\} + X_{d_j}^{c_k} \end{cases}$$

where $d_j \in DDG \wedge d_{i+1} \to d_j \to d_{i'}$, $c_k \in \{c_1, c_2, ...c_m\}$.

By using the above iterative function, we can calculate the values of vertices that are needed for calculating the weight of $e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}>$, as shown in Figure 5. Based on Definition 6, the minimum regeneration cost of d_j with the stored provenance d_i is the minimum value of vertices in V_{d_j}, formally[7]

$$minGenCost(d_j) = \min_{h=1}^{m} \left\{ ver_{d_j}^{c_h} \right\}.$$

Figure 5 demonstrates the iteration process of calculating the minimum regeneration costs of datasets in $\left\{ d_k \middle| d_k \in DDG \wedge d_i \to d_k \to d_j \right\}$, which are needed for calculating the weight of $e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}>$.

Based on this iterative method, we can further derive the weight of the edge

$$e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}> = Y_{d_{i'}}^{c_{s'}} + \sum_{\{d_k | d_k \in DDG \wedge d_i \to d_k \to d_{i'}\}} \left(minGenCost(d_k) * v_{d_k} \right) = Y_{d_{i'}}^{c_{s'}} + \sum_{\{d_k | d_k \in DDG \wedge d_i \to d_k \to d_{i'}\}} \left(\min_{h=1}^{m} \left\{ ver_{d_k}^{c_h} \right\} * v_{d_k} \right).$$

This iterative method is a general method for calculating the weights of edges in the CTG. However, in today's market, most cloud service providers do not charge data inbound cost (e.g. Amazon, Microsoft, etc.). This means that the costs of transferring a dataset from one place to all other cloud service providers are the same. Hence, in order to regenerate d_{i+1}, the only two options are either 1) keeping d_i in c_s where d_i is resided or 2) transferring d_i to the cloud service provider which has the lowest computation price, denoted as c_l, to regenerate d_{i+1}.

By introducing this condition, we can simplify the iteration process. To calculate the weight of $e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}>$, we only need the values of vertices in the set of $\left\{ ver_{d_j}^{c_s}, ver_{d_j}^{c_l} \middle| d_j \in \mathrm{DDG} \wedge (d_i \to d_j \to d_{i'}) \right\}$, which can be calculated as follows

$$\begin{cases} ver_{d_{i+1}}^{c_k} = Z_{d_i}^{c_s,c_k} + X_{d_{i+1}}^{c_k} \\ ver_{d_j}^{c_k} = \min \left\{ ver_{d_{j-1}}^{c_k}, (ver_{d_{j-1}}^{c_s} + Z_{d_{j-1}}^{c_s,c_k}) \right\} + X_{d_j}^{c_k} \end{cases}$$

where $d_j \in DDG \wedge d_{i+1} \to d_j \to d_{i'}$, $c_k \in \{c_s, c_l\}$.

By using the above new iterative function, we can calculate the values of vertices that are needed for calculating the weight of $e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}>$, and the minimum regeneration cost of d_j with the stored provenance d_i is the smaller value of the two vertices $ver_{d_j}^{c_s}$ and $ver_{d_j}^{c_l}$, formally

$$minGenCost(d_j) = \min \left\{ ver_{d_j}^{c_s}, ver_{d_j}^{c_l} \right\}.$$

Based on this iterative method, we can further derive the weight of the edge

$$e<ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}}> = Y_{d_{i'}}^{c_{s'}} + \sum_{\{d_k | d_k \in DDG \wedge d_i \to d_k \to d_{i'}\}} \left(minGenCost(d_k) * v_{d_k} \right) = Y_{d_{i'}}^{c_{s'}} + \sum_{\{d_k | d_k \in DDG \wedge d_i \to d_k \to d_{i'}\}} \left(\min \left\{ ver_{d_k}^{c_s}, ver_{d_k}^{c_l} \right\} * v_{d_k} \right).$$

Lemma 2: *The length of every path from ver_{start} to ver_{end} in the CTG equals to the total cost rate of the DDG with the corresponding storage strategy and the minimum cost regeneration strategy for the deleted datasets.*

Proof: A path from ver_{start} to ver_{end} in the CTG is composed of connected edges, and its length equals the sum of weights of these edges. According to Definition 5, the start and end vertices of every edge are deemed as stored datasets where their storage cost rates are added to the weights of edges. Also the datasets that the edges cross over are deemed as deleted datasets where their minimum regeneration cost rates are added to the weights of edges. Hence, the length of a path

[7] The formula of minimum regeneration cost naturally holds based on Definition 6 and the iteration process, hence we need not give a formal proof for this formula.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

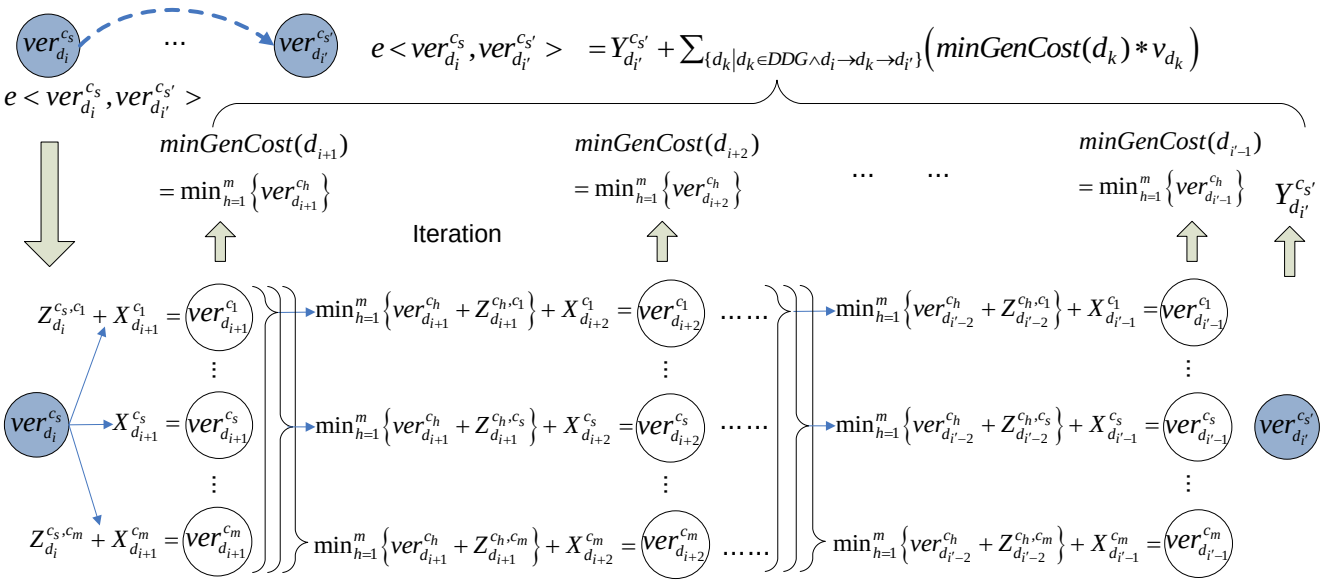


Figure 5. Iteration process for calculating the weight of an edge in CTG

from ver_{start} to ver_{end} contains 1) the sum of storage cost rates of datasets that the path traverses and 2) the sum of minimum regeneration cost rates of datasets that the path crosses over, which is the total cost rate of the storage strategy that the path represents. **Lemma 2 holds.**

Step 3: Find the shortest path of the CTG.

From the above construction steps, we can clearly see that the CTG is a weighted directed acyclic graph. Hence we can use the Dijkstra algorithm to find the shortest path from ver_{start} to ver_{end}. The Dijkstra algorithm is a classic greedy algorithm to find the shortest path in graph theory. We denote the shortest path from ver_{start} to ver_{end} as $P_{\min} < ver_{start}, ver_{end} >$.

Based on the above steps of the GT-CSB algorithm, we can draw the following theorem.

Theorem: *Given a linear DDG with datasets {d_1, d_2 … d_n} and m cloud services {c_1, c_2 … c_m} for storage, the length of* $P_{\min} < ver_{start}, ver_{end} >$ *of its CTG is the minimum cost rate for storing and regenerating datasets of the DDG in clouds, which is the minimum cost benchmark.*

Proof: If the length of $P_{\min} < ver_{start}, ver_{end} >$ is not the minimum cost benchmark, then there exists another storage and regeneration strategy of the DDG, which has a smaller total cost rate. According to Lemma 1, we can find a path in the CTG that represents this strategy. According to Lemma 2, the length of the new path equals the total cost rate of the new strategy, which is smaller than the length of $P_{\min} < ver_{start}, ver_{end} >$. This is contradicting to the condition that $P_{\min} < ver_{start}, ver_{end} >$ is the shortest path from ver_{start} to ver_{end} in the CTG. Hence, the length of $P_{\min} < ver_{start}, ver_{end} >$ is the minimum cost benchmark of storing and regenerating dataset of the DDG in clouds. **Theorem holds.**

4.3 Computation Complexity Analysis

Based on the discussion in Section 4.2, we demonstrate the pseudo code of our GT-CSB algorithm in Figure 6.

From Figure 6 we can see that the algorithm Step 1 is demonstrated in lines 1-7. For a linear DDG with n datasets and m cloud service providers, we need to create mn vertices in the CTG in Step 1.1 (lines 1-4), and the number of edges added to the CTG in Step 1.2 (lines 5-7) is in the magnitude of m^2n^2.

Step 2 is demonstrated in lines 8-16, which includes the iteration process. To calculate the weight of an edge in the CTG, we need to calculate at most mn vertices' values (lines 11-12). Because of the iteration process, the calculated vertices' values can be reused in the next iteration round. Hence the time complexity for calculating the value of one vertex is $O(m)$ (line 13). Hence, the time complexity of calculating the weight of an edge is $O(m^2n)$. Hence, the time complexity of calculating weights of all edges in the CTG is $O(m^4n^3)$.

In Step 3 (line 17), the time complexity of Dijkstra algorithm is $O(m^2n^2)$.

Based on the above analysis, the total time complexity of the GT-CSB algorithm is $O(m^4n^3)$. The space complexity of the GT-CSB algorithm is $O(m^2n^2)$, which is the space to save the created CTG.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

Algorithm: G**T-CSB**
Input: A linear DDG $\{d_1, d_2 \dots d_n\}$;
Cloud service providers $\{c_1, c_2 \dots c_m\}$;
Output: The minimum cost benchmark

01. Create ver_{start}, ver_{end};
02. **for** (every dataset d_i in DDG) //Create vertices for CTG
03. Create $V_{d_i} = \{ver_{d_i}^{c_1}, ver_{d_i}^{c_2}, ..., ver_{d_i}^{c_m}\}$;
04. Add vertices $V_{d_1} \bigcup V_{d_2} ... \bigcup V_{d_n} \bigcup \{ver_{start}, ver_{end}\}$ to CTG;
05. **for** (every $ver_{d_i}^{c_s} \in$ CTG) //Create edges for CTG
06. **for** (every $ver_{d_{i'}}^{c_{s'}} \in \text{CTG} \wedge (d_i, d_{i'} \in \text{DDG} \wedge d_i \rightarrow d_{i'})$)
07. Create $e < ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}} >$; //Create an edge
08. **for** (every $ver_{d_{i+1}}^{c_k} \in V_{d_{i+1}}$) //Calculate the edge weight
09. $ver_{d_{i+1}}^{c_k} = Z_{d_i}^{c_s, c_k} + X_{d_{i+1}}^{c_k}$;
10. $weight = \min_{h=1}^{m}\{ver_{d_{i+1}}^{c_h}\}$;
11. **for** (every $d_j \in \text{DDG} \wedge (d_{i+1} \rightarrow d_j \rightarrow d_{i'})$)
12. **for** (every $ver_{d_j}^{c_k} \in V_{d_j}$)
13. $ver_{d_j}^{c_k} = \min_{h=1}^{m}\left\{ver_{d_{j-1}}^{c_h} + Z_{d_{j-1}}^{c_h, c_k}\right\} + X_{d_j}^{c_k}$;
14. $weight = weight + \min_{h=1}^{m}\{ver_{d_j}^{c_h}\}$;
15. $weight = weight + Y_{d_{i'}}^{c_s, c_{s'}}$;
16. Set $e < ver_{d_i}^{c_s}, ver_{d_{i'}}^{c_{s'}} > = weight$; //Set edge weight
17. P = Dijkstra (ver_{start}, ver_{end}, CTG); //Find the shortest path
18. Return P; //The benchmark is the length of P

Figure 6. Pseudo code of GT-CSB algorithm

5 Utilisation of GT-CSB Algorithm

The major contribution of the GT-CSB algorithm is to calculate the minimum cost for storing and regenerating datasets in multiple clouds. It can be widely utilised in cloud computing. As stated in Section 1, by using this algorithm, users can 1) design minimum cost benchmarking approaches to evaluate the cost effectiveness in clouds; 2) design cost-effective strategies to store and regenerate application datasets and 3) understand the cost constitution of their application. In our prior work [20] [28] [21], we proposed the CTT-SP algorithm that can find the best trade-off between compuatation and storage in one cloud, and used it in designing cost-effective storage strategies and benchmarking approaches. As the same philosophy can be applied in this paper, we briefly introduce how to design minimum cost benchmarking approaches and cost-effective strategies based on the GT-CSB algorithm for storing and regenerating datasets in clouds. In Section 6, we will demonstrate the constitution of different costs for the application under different types of workload.

5.1 Minimum Cost Benchmarking Approaches

In this section, we describe how to facilitate benchmarking approaches by using the GT-CSB algorithm. According to different usage scenarios, we have two different benchmarking approaches.

5.1.1 Static On-Demand Minimum Cost Benchmarking

In clouds, whenever users want to know the minimum cost benchmark, they can launch this approach on the DDG of all application datasets and wait for the benchmark to be calculated. This approach is more suitable for the situation that less frequent benchmarking is requested, primarily before runtime. In order to build the approach, we need to apply the GT-CSB algorithm to the general DDG. In our prior work [20], we proposed a recursive algorithm for general DDG to find the best trade-off between computation and storage costs in one cloud, which is based on the CTT-SP algorithm. The same procedure can be adapted to the GT-CSB algorithm, based on which we can build a static on-demand minimum cost benchmarking approach. Specifically, given a general DDG, we randomly choose one linear data dependency path as "main branch" to construct the CTG, and the rest of datasets in the DDG are deemed as "sub-branch". Then, we can call the GT-CSB algorithm on the CTG of the "main branch", and recursively call the algorithm for the "sub-branch" based on smart rules for setting the weights to different types of edges. Finally, we can find the minimum cost storage and regeneration strategy of the whole DDG.

5.1.2 Dynamic On-the-fly Minimum Cost Benchmarking

Although the general GT-CSB algorithm can calculate the minimum cost benchmark, its computation complexity is high. If there are frequent benchmarking requests, the on-demand approach will be inefficient, because we have to run the genreal GT-CSB on the whole DDG for every request. Hence, we need another approach which is more suitable for the situation that more frequent benchmarking is requested at runtime. In our prior work [28], we also proposed a dynamic on-the-fly benchmarking approach for one cloud by taking advantage of saved pre-calculated results. In this approach, we utilise the classic Master-Worker architecture in the implementation which guarantees the efficiency of pre-calculation. The same philosophy can also be applied in the GT-CSB algorithm to achieve a dynamic on-the-fly minimum cost benchmarking approach. Specifically, for every linear segment of a general DDG, we pre-calculate all its possible minimum cost storage and regeneration strategies and save them in a solution space. By merging and utilising the saved solution spaces, we can derive the minimum cost benchmark of the whole DDG and dynamically keep it updated whenever new datasets are generated or existing datasets' usage frequencies are changed.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

5.2 Cost-Effective Storage and Regeneration Strategy

The GT-CSB algorithm can also be used for designing cost-effective storage and regeneration strategies. Different from the benchmarking approach, the minimum cost strategy may not be the best strategy for storing and regenerating datasets. Besides cost effectiveness, a good strategy should take users' preferences into consideration (e.g., users' tolerance of data accessing delay). In our prior work [21], we developed a local-optimisation based strategy for storing and regeneration datasets in one cloud, which is highly cost-effective and practical. The same philosophy can be applied in this work to derive our cost-effective storage and regeneration strategy with multiple cloud services.

Specifically, in order to reflect users' preferences, we can enhance the GT-CSB algorithm by introducing two parameters denoted as T *and* λ, which are same as our prior work [21]. T is the parameter used to represent users' tolerance on data accessing delay. Users need to inform the cloud service provider about the datasets that they have requirements on their availabilities. λ is the parameter used to adjust the storage strategy when users have extra budget on top of the minimum cost benchmark to store more datasets for reducing the average datasets accessing time. Based on the enhanced GT-CSB algorithm, we can adapt the philosophy of local-optimisation to achieve the high efficiency of the storage strategy, which is as follows:

(1) Given a general DDG, we first partition it into linear segments and apply the GT-CSB algorithm to calculate the storage strategy.

We search for the datasets that have multiple direct predecessors or successors (i.e. the join and split datasets in the DDG), and use these datasets as the partitioning points to divide it into linear DDG segments, as shown in Figure 7. Based on the linear DDG segments, we use the GT-CSB algorithm to find their storage strategies. This is the essence for achieving the cost-effectiveness.

(2) When new datasets are generated in the system, they are treated as a new DDG segment and added to the old DDG. Correspondingly, its storage status is calculated in the same way as the old DDG.

(3) When a dataset's usage frequency is changed, the storage status of the linear DDG segment that contains this dataset is re-calculated.

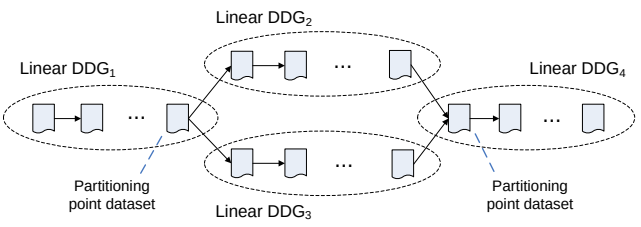


Figure 7. Dividing a DDG into linear DDG segments

By utilising the GT-CSB algorithm, our strategy achieves the local-optimisation of storing datasets in the DDG. The theoretical analysis of cost-effectiveness for the local-optimisation based strategy is given in our prior work [21]. In Section 6, we will demonstrate experimental results to further evaluate the cost-effectiveness of our strategy.

6 Evaluation

As Amazon is a well-known and widely recognised cloud service provider, we conduct experiments on Amazon cloud using on-demand services for simulation. We implement the GT-CSB in the Java programming language and run it on the virtualised EC2 instance with the Amazon Linux Image to evaluate its cost effectiveness and efficiency. We choose the standard small instance (m1.small) to conduct the experiments, because it is the basic type of EC2 CPU instances, which has a stable performance of one ECU[8]. In the simulation, we use randomly generated DDG with datasets of random sizes, generation times and usage frequencies. We also use popular cloud service providers' pricing model. The experiment code is available at http://www.ict.swin.edu.au/personal/yyang/doc/TCC14.zip. In this section, we summarise the evaluation results.

In Section 6.1, we evaluate the cost effectiveness of the minimum cost benchmark. We compare it with different representative storage strategies and demonstrate their cost differences to the minimum cost benchmark. In Section 6.2, we investigate the cost constitution in the minimum cost benchmark. We demonstrate the proportions of computation, storage and bandwidth costs in the minimum cost benchmark and the change of the proportion with different DDG inputs. In Section 6.3, we evaluate the efficiency of the GT-CSB algorithm. We demonstrate that the algorithm has a polynomial time complexity as both the number of datasets in the DDG and the number of cloud service providers grow.

6.1 Cost Effectiveness Evaluation

We evaluate the cost effectiveness of our minimum cost benchmark by comparing it with some representative storage strategies as follows.

- Store all datasets strategy, in which all generated datasets of the application are stored in the cloud. This strategy represents the common approach used in most applications in the cloud.
- Store none datasets strategy, in which all generated datasets of the application are deleted after being used. This strategy is often used in scientific applications that generate large and rarely used intermediate datasets.
- Cost rate based strategy reported in [29] [30], in which we store datasets in the cloud by comparing their own generation cost rate and storage cost rate.

[8] ECU (EC2 Computing Unit) is the basic unit defined by Amazon to measure the compute resources. Please refer to the following address for details: http://aws.amazon.com/ec2/instance-types/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

- Local-optimisation based strategy reported in [31] [21], in which we only achieve the localised optimum of the trade-off between computation and storage in the cloud.
- T-CSB algorithm based strategy reported in [26], in which datasets can be transferred to different cloud service providers for storage.

These strategies are designed for deploying applications with one cloud service provider, which is assumed to be Amazon cloud, i.e. using EC2 service ($0.10 per CPU instance hour) for computation and S3 service ($0.15 per gigabyte per month) for storage. In the T-CSB algorithm based strategy, we assume that datasets can be transferred to Haylix cloud for storage.

Haylix is a leading Australian IaaS cloud service provider, who provides reliable cloud storage with fast access for local Australian users. As data transfer over the Internet is often expensive and relatively slow in general, some cloud service providers (e.g. Amazon) cooperate with network infrastructure providers (e.g. Equinix) to provider dedicate connection service (e.g. AWS Direct Connect) for boosting the data transfer speed in and out of the cloud. Hence, we use the pricing models of Haylix and AWS Direct Connect in our simulation, i.e. $ 0.11 per CPU instance hour for computation, $0.12 per gigabyte per month for storage, $0.046 per gigabyte for outbound data transfer from Haylix.

In real world applications (e.g., case studies in Section 3 and supplementary materials), generated datasets vary dramatically in terms of size, generation time, usage frequency and the sturcutre of DDG. Hence, we set random parameters in our experiments to evaluate the general performance of our GT-CSB algorithm without the loss of generality. Specifically, we randomly generate DDGs with different number of datasets, each with a random size from 1GB to 100GB. The generation time is also random, from 10 hours to 100 hours. The usage frequency is again random, from once per month to once per year.

The simulation results are demonstrated in Figure 8. As we can see that the "store all datasets" and "store none dataset" strategies are very cost ineffective. By investigating the trade-off between computation and storage, the "cost rate based strategy" and "local-optimisation based strategy" can smartly choose to store or delete the datasets, thereby largely reducing the cost rate for storing datasets with one cloud service provider. If more cloud storage services are available, the simulation of "T-CSB algorithm based strategy with Haylix storage" demonstrates further reduction of the cost rate by taking bandwidth cost into account. However, it cannot achieve the minimum cost due to the limitation of the T-CSB algorithm, i.e., computation can only happen in one cloud. In contrast, our GT-CSB algorithm based minimum cost benchmark has the lowest cost rate among different storage strategies.

6.2 Constitution of Differnet Costs in Clouds

One of the important utilisations of the minimum cost benchmark is to help users to understand the constitution of different costs of their applications in the clouds. In this sub-section, we demonstrate the constitution of computation, storage and bandwidth costs in the minimum cost benchmark of different input DDGs.

We use a randomly generated DDG with 50 datasets and 3 cloud service providers with the pricing models as follows.

- Cloud 1: $0.11 per hour CPU, $0.1 per gigabyte per month for storage, and $0.01 per gigabyte for outbound data transfer.
- Cloud 2: $0.15 per hour CPU, $0.05 per gigabyte per month for storage, and $0.15 per gigabyte for outbound data transfer.
- Cloud 3: $0.12 per hour CPU, $0.07 per gigabyte per month for storage, and $0.03 per gigabyte for outbound data transfer.

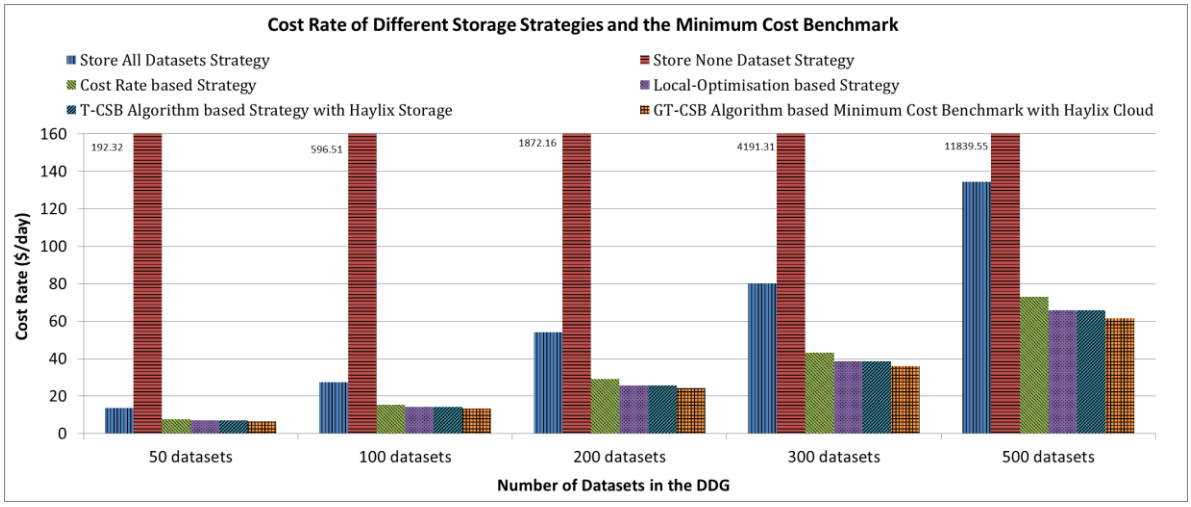


Figure 8. Cost-effectiveness comparison

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2491920, IEEE Transactions on Cloud Computing

We only use the above prices as representatives, as many cloud service providers (e.g. GoGrid[9], Rackspace[10], Haylix[11], etc.) have similar pricing models.

Based on the above setting, we calculate the minimum cost benchmark for a 50 datasets DDG with 3 cloud service providers. Figure 9 demonstrates the constitution of costs in this benchmark and how datasets are stored and regenerated in the three cloud service providers.

Given different DDGs, the constitution of costs in the benchmark will be different. Next, we change the parameters for generating the DDGs (i.e. size, regeneration time and usage frequency), and demonstrate the corresponding change of the costs constitution in the minimum cost benchmark, as well as the change of storage and regeneration strategies in the three cloud service providers.

Figures 10, 11 and 12 show the impact of datasets' sizes, regeneration times and usage frequencies on the minimum cost benchmark respectively. We can see from the figures that when datasets' sizes, regeneration times or usage frequencies are doubled or halved in the DDG, the constitution of costs in the benchmark does not change proportionally. Understanding the relationship between DDG and the constitution of costs in the minimum cost benchmark can help users to optimise the cost of their applications in clouds.

6.3 Efficiency Evaluation of the GT-CSB Algorithm

As presented in Section 4.3, the GT-CSB algorithm has a polynomial time complexity of $O(m^4n^3)$. In this sub-section we demonstrate the algorithm efficient of our implementation. Figure 13 (a) shows the results of running the GT-CSB algorithm on three cloud service providers with different number of datasets in the DDG. Figure 13 (b) shows the results of running the GT-CSB algorithm on a 50 datasets DDG with different number of cloud service providers. As we can see from Figure 13, the CPU time of executing the GT-CSB algorithm has a polynomial growth as the increase of datasets in the DDG or number of cloud service providers.

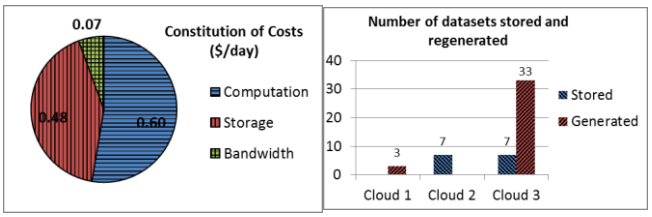
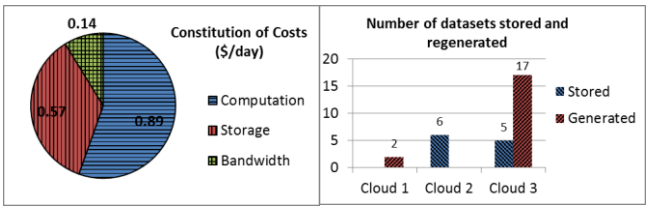
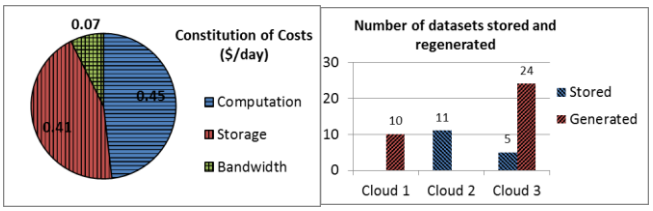


Figure 9. Constitution of costs in the minimum cost benchmark

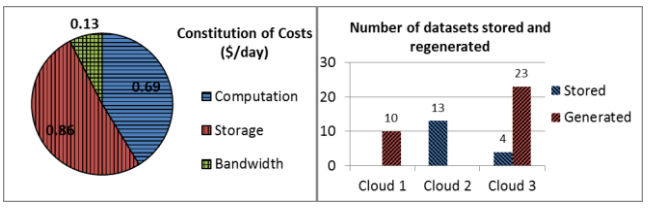


(a) Datasets' sizes are doubled in the DDG

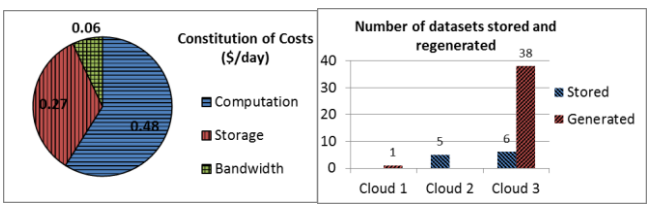


(b) Datasets' sizes are halved in the DDG

Figure 10. Impact of datasets' sizes on the minimum cost benchmark

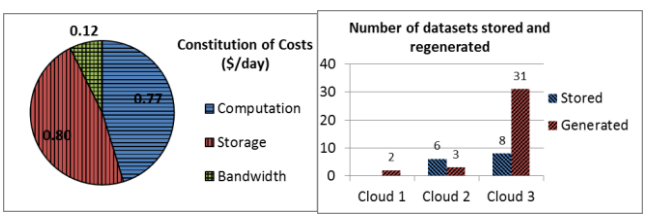


(a) Datasets' regeneration times are doubled in the DDG

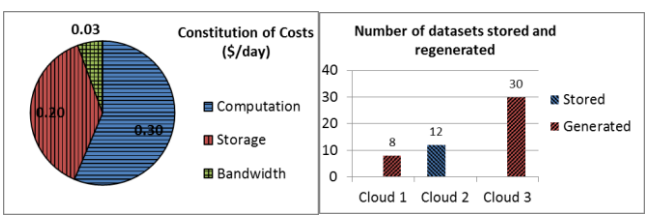


(b) Datasets' regeneration times are halved in the DDG

Figure 11. Impact of datasets' regeneration times on the minimum cost benchmark



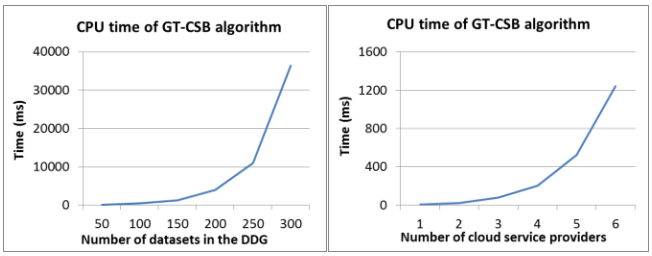
(a) Datasets' usage frequencies are doubled in the DDG



(b) Datasets' usage frequencies are halved in the DDG

Figure 12. Impact of datasets' usage frequencies on the minimum cost benchmark

[9] GoGrid: http://www.gogrid.com/
[10] Rackspace: http://www.rackspace.com/
[11] Haylix: http://www.haylix.com/



(a) 3 cloud service providers (b) 50 datasets DDG

Figure 13. Efficiency evaluation of GT-CSB algorithm

7 Conclusions and Future Work

In this paper, we have investigated the minimum cost strategy for storing and regenerating datasets based on the pay-as-you-go model with multiple cloud service providers. Towards achieving a generic best trade-off among computation, storage and bandwidth, we have designed the GT-CSB algorithm, which calculates the minimum cost benchmark for storing and regenerating datasets in clouds. We have presented the design of the algorithm in detail and rigid proof to guarantee the validity of minimum cost benchmark. Experimental results also demonstrated the excellent performance of the proposed approach.

In our current work, we assume that cloud service providers have unified prices for computation, storage and bandwidth resources. However, in the real world, the prices of cloud services can well be different according to different requirements and usages. Furthermore, extra cost might be caused by the "vender lock-in" issue among different cloud service providers, large number of requests from input/output (I/O) internsive applications, etc. In the future, we will incorporate more complex pricing models in our datasets storage and regeneration cost model.

In this paper, our focus is to solve the crucial problem of calculating the minimum cost for data storage and regeneration in multiple clouds. Hence the efficiency of the GT-CSB algorithm design has not been comprehensively investigated. In the future, we will re-design the algorithm by using Dynamic Programming techniques, which can further significantly reduce the algorithm complexity.

Acknowledgment

The research work reported here is partly supported by Australian Research Council under DP110101340 and LP130100324.

References

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

[2] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," *IEEE Transactions on Cloud Computing*, vol. 2, pp. 14-28, 01/01 2014.

[3] *Amazon Cloud Services*. Available: http://aws.amazon.com/

[4] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, *et al.*, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice and Experience*, pp. 1039–1065, 2005.

[5] R. Bose and J. Frew, "Lineage Retrieval for Scientific Data Processing: A Survey," *ACM Computing Surveys*, vol. 37, pp. 1-28, 2005.

[6] S. Agarwala, D. Jadav, and L. A. Bathen, "iCostale: Adaptive Cost Optimization for Storage Clouds," in *IEEE International Conference on Cloud Computing (CLOUD2011)*, 2011, pp. 436-443.

[7] H. Xu and B. Li, "Dynamic Cloud Pricing for Revenue Maximization," *IEEE Transactions on Cloud Computing*, vol. 1, pp. 158-171, 07/01 2013.

[8] K. Deng, J. Song, K. Ren, D. Yuan, and J. Chen, "Graph-Cut Based Coscheduling Strategy Towards Efficient Execution of Scientific Workflows in Collaborative Cloud Environments," in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, 2011, pp. 34-41.

[9] W. Li, Y. Yang, J. Chen, and D. Yuan, "A Cost-Effective Mechanism for Cloud Data Reliability Management based on Proactive Replica Checking," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, 2012, pp. 564-571.

[10] K. Deng, L. Kong, J. Song, K. Ren, and D. Yuan, "A Weighted K-Means Clustering Based Co-scheduling Strategy towards Efficient Execution of Scientific Workflows in Collaborative Cloud Environments," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)* 2011, pp. 547-554.

[11] P. Chen, B. Plale, and M. S. Aktas, "Temporal representation for scientific data provenance," in *8th International Conference on E-Science (e-Science2012)*, 2012, pp. 1-8.

[12] L. J. Osterweil, L. A. Clarke, A. M. Ellison, R. Podorozhny, A. Wise, E. Boose, *et al.*, "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.

[13] I. Foster, J. Vockler, M. Wilde, and Z. Yong, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," in *14th International Conference on Scientific and Statistical Database Management, (SSDBM'02)*, Edinburgh, Scotland, UK, 2002, pp. 37-46.

[14] K.-K. Muniswamy-Reddy, P. Macko, and M. Seltzer, "Provenance for the Cloud," in *8th USENIX Conference on File and Storage Technology (FAST'10)*, San Jose, CA, USA, 2010, pp. 197-210.

[15] J. Zhan, L. Wang, X. Li, W. Shi, C. Weng, W. Zhang, *et al.*, "Cost-aware Cooperative Resource Provisioning for Heterogeneous Workloads in Data Centers," *IEEE Transactions on Computers*, vol. 62, pp. 2155-2168, 2013.

[16] R. Chen, X. Weng, B. He, M. Yang, B. Choi, and X. Li, "Improving Large Graph Processing on Partitioned Graphs in the Cloud," in *ACM Symposium on Cloud Computing (SoCC2012)*,

2012.

[17] P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang, "Nectar: Automatic Management of Data and Computation in Datacenters," in *9th Symposium on Operating Systems Design and Implementation (OSDI'2010)*, Vancouver, BC, Canada, 2010, pp. 1-14.

[18] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The Cost of Doing Science on the Cloud: the Montage Example," in *ACM/IEEE Conference on Supercomputing (SC'08)*, Austin, Texas, 2008, pp. 1-12.

[19] I. Adams, D. D. E. Long, E. L. Miller, S. Pasupathy, and M. W. Storer, "Maximizing Efficiency by Trading Storage for Computation," in *Workshop on Hot Topics in Cloud Computing (HotCloud'09)*, San Diego, CA, 2009, pp. 1-5.

[20] D. Yuan, Y. Yang, X. Liu, and J. Chen, "On-demand Minimum Cost Benchmarking for Intermediate Datasets Storage in Scientific Cloud Workflow Systems," *Journal of Parallel and Distributed Computing*, vol. 71, pp. 316-332, 2011.

[21] D. Yuan, Y. Yang, X. Liu, W. Li, L. Cui, M. Xu, *et al.*, "A Highly Practical Approach towards Achieving Minimum Datasets Storage Cost in the Cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 1234-1244, 2012.

[22] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multi-Resource Allocation: Fairness-Efficiency Tradeoffs in a Unifying Framework," in *2012 Proceedings IEEE INFOCOM 2012*, 2012, pp. 1206-1214.

[23] J. Baliga, R. W. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, pp. 149-167, 2011.

[24] F. Chen, J. Grundy, Y. Yang, J.-G. Schneider, and Q. He, "Experimental Analysis of Task-Based Energy Consumption in Cloud Computing Systems," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, 2013, pp. 295-306.

[25] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "Spanstore: Cost-Effective Geo-Replicated Storage Spanning Multiple Cloud Services," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP 2013)*, 2013, pp. 292-308.

[26] D. Yuan, X. Liu, L. Cui, T. Zhang, W. Li, D. Cao, *et al.*, "An Algorithm for Cost-Effectively Storing Scientific Datasets with Multiple Service Providers in the Cloud," in *9th International Conference on e-Science (e-Science2013)*, , 2013, pp. 285-292.

[27] S. Xu, J. H. Beynon, D. Ruan, and G. Lu, "Experimental study of the out-of-plane dynamic compression of hexagonal honeycombs," *Composite Structures*, vol. 94, pp. 2326-2336, 2012.

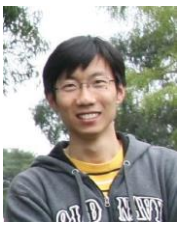
[28] D. Yuan, X. Liu, and Y. Yang, "Dynamic on-the-fly Minimum Cost Benchmarking for Storing Generated Scientific Datasets in the Cloud," *IEEE Transactions on Computers*, vol. 64, pp. 2781-2795, 2015.

[29] D. Yuan, Y. Yang, X. Liu, and 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.

[30] D. Yuan, Y. Yang, X. Liu, G. Zhang, and J. Chen, "A Data Dependency Based Strategy for Intermediate Data Storage in Scientific Cloud Workflow Systems," *Concurrency and Computation: Practice and Experience*, vol. 24, pp. 956-976, 2010.

[31] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A Local-Optimisation based Strategy for Cost-Effective Datasets Storage of Scientific Applications in the Cloud," in *Proc. of 4th IEEE International Conference on Cloud Computing (Cloud2011)*, Washington DC, USA, 2011, pp. 179-186.

Authors:



Dong Yuan received the PhD degree from Swinburne University of Technology, Australia, in 2012. He is a lecturer at Sydney University. His research interests include cloud computing, data management in parallel and distributed systems, scheduling and resource management, business process management and workflow systems.

Lizhen Cui received the PhD degree from Shandong University in 2005. He is a full professor at Shandong University. His research interests include workflow and distributed data management for cloud computing, service computing.

Wenhao Li received the PhD from Swinburne University of Technology, Australia in 2014. He is a post-doc at Shandong University. His research interests include parallel and distributed computing, cloud and grid computing, workflow technologies and data management in distributed computing environment.

Xiao Liu received the PhD degree from Swinburne University of Technology, Australia in 2011. He is an associate professor at East China Normal University. His research interests include workflow management systems, scientific workflow, business process management and data mining.

Yun Yang received the PhD degree from the University of Queensland, Australia in 1992. He is a full professor at Swinburne University of Technology. His research interests include software technologies, cloud computing, workflow systems and service-oriented computing.