Journal of Computers Vol. 29 No. 1, 2018, pp. 166-174
doi:10.3966/199115992018012901015

A Greedy Approach with New Cost Model for Intermediate Datasets Storage Problem in General Workflows

Zimao Li[1], Yingying Wang[1*]

[1] College of Computer Science, South-Central University for Nationalities, Wuhan, China
lizm@mail.scuec.edu.cn, 2279457429@qq.com

Received 12 October 2016; Revised 6 March 2017; Accepted 30 March 2017

Abstract. Running a scientific workflow on the cloud will generate a large volume of intermediate datasets and many of them have valuable information that can be used for further study, but the cost of storing them all is unbelievably high for the enormous data size. A feasible solution is to keep some of the intermediate datasets stored and re-compute the others when needed, the intermediate dataset storage problem asks to find a tradeoff to minimize the total cost of storing or re-generating each of the intermediate datasets. This paper focuses on a new cost model for the problem with general workflow, which incorporates additional delay tolerance, usage frequency and the transfer cost to make the cost model becoming more general. Based on a directed acyclic graph describing the dependence relationship between datasets, a greedy approach for the problem is proposed and implemented. Experimental results demonstrate the effectiveness and efficiency of our algorithm.

Keywords: delay tolerance, greedy algorithm, intermediate datasets storage, transfer cost, usage rate

1 Introduction

A large volume of intermediate datasets will be generated when we run a scientific workflow. Some intermediate datasets may need to be stored for future use because scientists may need to re-analyze or share the intermediate results with professionals all over the world. The scientific analyses are usually execution intensive, hence taking a long time for execution, especially in fields such as astronomy [1], high-energy physics [2] and bio-informatics [3], generally terabytes of data have to be analyzed.

Given the large sizes of the datasets, running scientific workflow applications usually require high-performance computing resources and massive storage [4], thus grid systems [2] are good choices for deploying scientific workflows for their 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.

As the latest computing paradigm in recent 10 years, the emergence of cloud computing technologies [5] offers a new way to develop scientific workflow systems. Cloud computing provides redundant, inexpensive and scalable resources on demands to users [6], and one major research topic of cloud computing is cost-effective strategies for storing intermediate datasets.

Foster et al. made a comprehensive comparison of grid computing and cloud computing [7]. 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 because cloud computing systems are composed of data centers which can be clusters of commodity hardware [5]. The work by Deelman et al. [8] shows that cloud computing offers a cost-effective solution for data-intensive applications, such as scientific workflows. 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 intermediate datasets generated during workflow executions may cause a high storage cost. In contrast, if we delete all the intermediate datasets and re-generate them every time they are needed, the

* Corresponding Author

computation cost of the system may well be very high too. A possible solution is to find a balance that selectively stores some popular datasets and regenerates the rest of them when needed, the *intermediate dataset storage problem* (IDSP for short) asks a decision for each intermediate dataset to be stored or re-generated to minimize the total cost of storage and re-computation.

Some strategies have already been proposed for IDSP in one cloud or multiple clouds. When the usage rate was taken into consideration in a single cloud, Yuan et al. introduced an intermediate datasets dependency graph [9] according to the data provenance [10-11] and proposed the minimum benchmark for the IDSP, an algorithm for linear IDSP with time complexity $O(n^4)$, as well as an algorithm for a parallel IDSP which takes $O(n^9)$ time. When both of usage rate and delay tolerance were considered in the cost model in a single cloud, Yuan et al. [12-13] presented dynamic strategies for the problem, which are feasible but not optimal solutions for general IDSP. In [12] they only updated the cost model with the additional delay tolerance in the foundation of [13]. Xu et al. [14] designed a feasible but not optimal genetic algorithm for IDSP and they only took the usage rate into consideration in one cloud. In 2015, Yuan et al. proposed the newest algorithm with time complexity $O(m^4n^3)$ for the linear IDSP in multiple clouds and gave an idea for the general workflow [6], they aimed to find the balance among the storage cost, computation cost and transfer cost but failure to consider additional factors associated with the costs. Cheng et al. proposed a binary-tree algorithm for the linear workflow in a cloud, which is the best possible at present for dealing with linear IDSP [15]. They did a profound theoretical discussion. In this paper, besides usage rate and delay tolerance, we introduce another factor called transfer cost into the model since cloud service providers generally place high cost on data transfer in and out their data centers [16] and propose a greedy algorithm for IDSP in multiple clouds with the more general cost model.

The rest of the paper is organized as follows. Section 2 defines different notations, notions and factors used in the general cost model. Section 3 describes the greedy algorithm and its analysis. The implementation results are demonstrated in Section 4. Section 5 concludes and perspects future work.

2 New Cost Model

We simplify a scientific workflow as $W = (D, L)$, where $D = \{d_0, d_1, d_2, ..., d_n\}$ is the set of intermediate datasets with input dataset d_0 and L represents the set of dataset dependences in the form $< d_i, d_j >$, d_i, d_j are two datasets in the workflow. d_i is called the direct predecessor of d_j, and d_j is the direct successor of d_i. We can construct a directed acyclic graph (DAG for short) to describe W, where the vertexes and directed arcs represent the datasets and the dependencies between them, respectively (See Fig. 1).

Given a cloud workflow, there are two options for a dataset d_i, stored and deleted. Generally, the storage cost of a dataset d_i is proportional to its size and the stored time, and the computation cost is proportional to its computation time. We use $s(d_i)$, $t_1(d_i)$ and $t_2(d_i)$ to denote the size, the computing time and the storing time of dataset d_i, respectively. Values of $s(d_i)$, $t_1(d_i)$ and $t_2(d_i)$ are given for each dataset d_i. We denote $x(d_i)$ as d_i's unit cost per unit size per unit time and $y(d_i)$ as d_i's unit cost per unit time. For convenience of analysis, we consider there is no difference of unit cost of storing in different clouds.

If a dataset d_i is selected to be stored, the storage cost for d_i is the product of $x(d_i)$, $s(d_i)$ and $t_2(d_i)$, that is,

$$CostS(d_i) = x(d_i) * s(d_i) * t_2(d_i) \tag{1}$$

If a dataset d_j is selected to be computed, the computation cost d_j is defined by formula (3).

It is natural to separate the set of dataset D into two sets $D = (S, C)$, where S and C represents the set of storage datasets and the set of computation datasets, respectively.

Definition 1. Given a computation dataset d_j of a workflow DAG, we call dataset d_i is the **storage-prior** of d_j, if and only if in the path from d_i to d_j, d_i is the only storage dataset and others are computation datasets. We call the path from the direct successor of d_i to d_j is the **computation path** of d_j, denoted as $ap(d_j)$. $AP(d_j)$ is used to denote the set of all computation paths of d_j. We also use $SP(d_j)$ to denote the set of all storage-priors of d_j. It is easy to find that generally $|SP(d_j)| = |AP(d_j)|$.

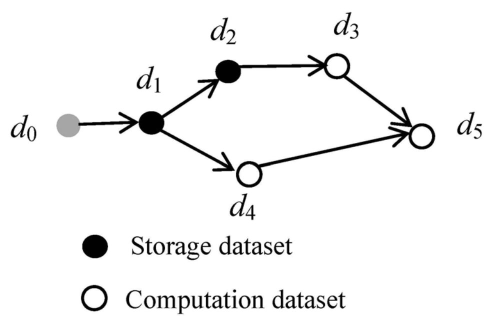

Fig. 1. An example of a workflow

For example, in Fig. 1, $SP(d_5) = \{d_1, d_2\}$, $ap(d_5)_1 = \{d_3, d_5\}$, $ap(d_5)_2 = \{d_4, d_5\}$, thus $AP(d_5) = \{ap(d_5)_1, ap(d_5)_2\}$. When we want to re-generate a computation dataset d_5, we will have to find its computing paths $AP(d_5) = \{ap(d_5)_1, ap(d_5)_2\}$. Thus the computation cost of the d_5 can be calculated as:

$$CostC(d_5) = \sum_{\alpha \in AP(d_5)} \sum_{d \in \alpha, d \neq d_5} y(d) * t_1(d) + y(d_5) * t_1(d_5)$$

Generally, for a computation dataset d_j, the generation cost of d_j in the $ap(d_j)_1$ can be calculated as

$$CostC(d_j)_1 = \sum_{d \in ap(d_j)_1} y(d) * t_1(d) \qquad \textbf{(2)}$$

Since there may be more than one computing path of d_j, thus the generation cost of d_j is：

$$CostC(d_j) = \sum_{\alpha \in AP(d_j)} \sum_{d \in \alpha, d \neq d_j} y(d) * t_1(d) + y(d_j) * t_1(d_j) \qquad \textbf{(3)}$$

For a dataset d_i, the cost of d_i is influenced by the usage frequency. It is a forecasting value from the dataset's usage history recorded in the system logs [16]. Here, we choose the peak usage times of workflow execution in the system cloud. It is reasonable to use the rate of usage times of d_i and usages times of all datasets at the peak times.

Definition 2. Let $count_i$ denote the peak usage times of d_i in the past workflow executions, the usage frequency $f(d_i)$ of dataset d_i can be defined as:

$$f(d_i) = \frac{count_i}{\sum_{k=1}^{n} count_k} \qquad \textbf{(4)}$$

Users delay tolerance influences the cost of a dataset and the users' tolerance of accessing delay may differ dramatically. Sometimes users may want the data to be available immediately, and sometimes they may not care about waiting for it to become available [17]. One user may have different degrees of delay tolerance for different datasets and different users may also have different degrees of delay tolerance for one dataset, and one user may even have different degrees of delay tolerance for a dataset in different time phases. Hence, a parameter to indicate users' delay tolerance should be considered and be set flexibly by the system manager based on users' preferences.

Let m denote the number of the users and u_{ik} (k =1, 2, ..., m) the delay tolerance of the kth user to the ith dataset. When the kth user does not need the ith dataset, the u_{ik} is set to the infinite. The others are values about delay time according to preferences of the users to the dataset or the practical experiences. u_{ik} is a forecasting value which is decided according to the personal experiences and the analysis of the user's historical records. The higher u_{ik} is, the more tolerant the users are.

Definition 3. The users' delay tolerance $g(d_i)$ is defined as:

$$g(d_i) = \min\{u_{i1}, u_{i2}, \cdots, u_{im}\} \qquad \textbf{(5)}$$

In the practical applications, if the computing time is larger than $g(d_i)$, then d_i has to be stored no matter how large d_i' s cost is. By comparison with Dong Yuan's definition in [12], the users' tolerance with more than one user is more common.

Now we can define the relative generation cost, which is used partly to decide whether to store or compute d_i.

Definition 4. The relative generation cost, *G_Cost*(d_i) of d_i is defined as below:

$$G\_Cost(d_i)=\begin{cases} \dfrac{x(d_i)}{f(d_i)}*s(d_i)*t_2(d_i)\ if\ t_1(d_i)>g(d_i)//Store \\ \min\begin{cases} \dfrac{x(d_i)}{f(d_i)}*s(d_i)*t_2(d_i)\ if\ t_1(d_i)>g(d_i)//Store \\ \sum\limits_{\alpha\in AP(d_i)}\sum\limits_{d\in\alpha,d\neq d_i} y(d)*t_1(d)+y(d_i)*t_1(d_i)//Compute \end{cases} \end{cases} \quad \textbf{(6)}$$

The formula above is the cost of the dataset d_i in different status. It can be used to verify whether the dataset should be stored or computed. When the storage cost is lower than the regeneration cost for one dataset, the intermediate dataset d_i is to be stored, otherwise the dataset d_i is to be computed. An exception is that if the computing time of some dataset is larger than its delay tolerance, then the dataset is stored directly.

As cloud computing is growing so fast, different cloud service providers are appearing. Cloud service providers place high cost on data transfer in and out their data centers [16]. In contrast, data transfer within one cloud service provider's data centers are usually free. In the past since we generally store data in a specified location or different data centers of the same cloud due to the limitations of hardware facility and limited size of datasets, almost no data transfer was required to consider [8, 16]. However, scientific applications in recent days have to be stored and run in a distributed manner for the increasing data sizes and to facilitate data sharing [18-19], data transfer is inevitable since some datasets are required to store in fixed locations.

Definition 5. For a dataset d_i and different cloud service providers, the transfer cost introduced by dataset d_i, *CostT*(d_i) is defined as:

$$CostT(d_i)=\begin{cases} transfer\_cost[d_i\cdot present][d_i\cdot next]\ if\ d_i\in S \\ \sum\limits_{\alpha\in AP(d_i)\bigcup SP(d_i)}\sum\limits_{d\ is\ in\ a,d\neq d_i}^{n} transfer\_cost[d_i\cdot present][d_i\cdot next] \\ +transfer\_cost[d_i\cdot present][d_i\cdot next] \end{cases} \quad \textbf{(7)}$$

Here, d_i*.present* means current location of d_i and d_i*.next* means the next location of d_i during the execution. The dynamical *present* and *next* are from the placement strategies. D. Yuan et al. proposed an effective placement strategy [20]. Because we don't discuss the placement strategy in this paper, we think they are given values during the execution. For example, the *transfer_cost*[*i*][*j*] means the transfer cost of a dataset from *i*th cloud to *j*th cloud, where *i* or *j* is the cloud service providers. When *i* = *j*, *transfer_cost*[*i*][*j*] = 0 which means the dataset is kept in *i*th cloud and will not move into others. *CostT* can be used to determine the location (cloud) of each dataset that should be stored.

Definition 6. The total cost of all the datasets of a workflow is:

$$total\_Cost(W)=\sum_{d\in D}(G\_Cost(d)+CostT(d)) \quad \textbf{(8)}$$

Based on the concepts and formulas mentioned above, the problem of dataset storage can be described as follows:

Input: A given workflow *W*; the storage cost per unit size per unit time; computation cost per time of each intermediate dataset of *D*; the size of each dataset and the storing time; the computing time of each dataset; the transfer_cost[*i*][*j*] for all possible *i* and *j*; the usage frequency of each dataset; the delay tolerance of each user for each dataset.

Output: The set of storage dataset *S* and the set of computation dataset *C*.

Objective: the total cost of *W* is minimized.

3 Algorithms Description

As we know, the greedy algorithm is a kind of simple and efficient procedure for searching feasible solutions, sometimes it even finds an optimal solution for given problem. In this section, we propose a greedy algorithm for the IDSP problem with above cost model on multiple clouds, the algorithm is called IDS_Greedy.

Algorithm: IDS_Greedy(D, L, F, U)

Input: a workflow $W(D, L)$, D is a set of intermediate datasets $\{d_0, d_1, d_2, ..., d_n\}$ and L is a set of 0-1 adjacency matrix where 1 means pairs $\{d_i, d_j\}$ exists and d_i is the direct predecessor of d_j; the storage cost per size per min of the intermediate dataset $\{x_1, x_2, ..., x_n\}$ and the size of each intermediate dataset $\{s_1, s_2, ..., s_n\}$ and the time of storing the dataset $\{t_2(d_1), t_2(d_2), ..., t_2(d_n)\}$; the computing cost per time of the intermediate dataset $\{y_1, y_2, ..., y_n\}$ and the time of each intermediate dataset $\{t_1(d_1), t_1(d_2), ..., t_1(d_n)\}$; the usage frequency of each intermediate dataset $\{f(d_1), f(d_2), ..., f(d_n)\}$; the delay tolerance of each user for each intermediate dataset $\{u_{11}, u_{12}, ..., u_{1m}, ..., u_{21}, u_{22}, ..., u_{2m}, ..., u_{n1}, u_{n2}, ..., u_{nm}\}$; the transfer_cost[$i$][$j$].

Output: the direct acyclic graph with each node is colored by either of gray, white and black indicating the status of each intermediate datasets and the total cost is minimum. (gray: the start node, white: the computing node, black the storing node)

```
Begin
    create the DAG(V,E) according to the W(D,L);
    Initialize DAG,V={$v_0$,$v_1$,$v_2$,...,$v_n$} corresponding to {$d_0$,$d_1$,$d_2$,...,$d_n$};
    E:={$e_1$,$e_2$,...,$e_{|L|}$} corresponding to the adjacency pairs in L;
    $d_0$.flag:='S';
    $v_0$.color:='gray';
    G_Cost($d_0$):=0;
    CostT($d_0$):=0;
    total_cost(W):=G_Cost($d_0$)+CostT($d_0$);
    visited($v_0$):=1;
for i:=1 to n then
        $d_i$.falg:='S';
        $v_i$.color='black';
        G_Cost($d_i$):=CostS($d_i$);
        visited($v_i$)=0 ;
for i:=0 to n
        if L[0][i]:=1
        T:={$d_i$};
    While T!=NULL
           for j:=0 to T.length
              if visited(T[j])==0 then
                 g(T[j])←min{$u_{T[j]1}$,...,$u_{T[j]m}$};
                 if $t_1$(T[j])>g(T[j]) then
                      $d_{T[j]}$.flag:='S';
                     $v_{T[j]}$.color:='gray';
                           G_cost($d_{T[j]}$):=CostS;
                           CostT($d_{T[j]}$):=transfer_cost[$d_{T[j]}$.present]
                           [$d_{T[j]}$.next];
                           total_cost(W):=G_cost($d_{T[j]}$)+CostT($d_{T[j]}$);
                           visited($v_{T[j]}$):=1;
                  else if CostC(T[j])<G_Cost(T[j]) then
                        $d_{T[j]}$.flag:='C';
                       $v_{T[j]}$.color:='white';
                             G_Cost($d_{T[j]}$):=CostC;
                           CostT($d_{T[j]}$):=
                       $\sum_{\alpha\in AP(d_j)\cup SP(d_j)}\sum_{d_j\, is\, in\, \alpha, d\neq d_j} transfer\_cost[d.present][d.next]$
                       $+transfer\_cost[d_j.present][d_j.next]$
                                 total_cost(W):=G_cost($d_{T[j]}$)+CostT($d_{T[j]}$);
                              visited($v_{T[j]}$):=1;
                          For every dataset $d_k$ in AP(T[j]) or SP(T[j])
                          except T[j]
                        $d_k$.present:=T[j].present;
                        T[j].present:=T[j].next;
                    else
                      $d_{T[j]}$.flag:='S';
                      $v_{T[j]}$.color:='gray';
                          G_Cost($d_{T[j]}$):=CostS;
                        CostT($d_{T[j]}$):= transfer_cost[$d_{T[j]}$.present]
                        [$d_{T[j]}$.next];
                        total_cost(W):=G_Cost($d_{T[j]}$)+CostT($d_{T[j]}$);
                      visited($v_{T[j]}$):=1;
                      T[j].present:=T[j].next;
            Remove T[j] from T;
        for l=0 to n do
            for r=0 to n  do
                if L[r][l]==1 && visited($d_r$)==0 then
                        Break;
            if r==n
                T:={$d_l$};
    print DAG and the total_cost(W);
```

In this algorithm, we utilized the greedy algorithm to solve IDSP. In the loop of the while, the max loop times are *n*. And when we traversal *L* to update *T*, the loop times are n. Also, we must deal with all the datasets. To addition, we should take the delay tolerance and the tansfer_cost related to m clouds into consideration. So, the complex time is O(n^4m^2).

4 Simulation and results of Algorithm

To evaluate the performance of the algorithm that we proposed, we take the simple workflow in Fig. 1 as the example of workflow structure and then conduct the simulating experiment with eight strategies together to compare the total cost of the system. The strategies are: (1) store all the intermediate datasets; (2) store none; (3) Store top 20% often used datasets; (4) Store top 40% often used datasets; (5) Store top 60% often used datasets; (6) Store top 20% generation datasets; (7) Store top 40% generation datasets; (8) Store top 60% generation datasets.

The number of the intermediate datasets is 5. The dataset size varies from *1GB* to *100GB*. The generation time is random from 1 *min* to 60 *min*s. The storage time is random from *1 time* to *10 times per month*. The users' delay tolerance is also random from 1 *min* to 60 *min*s. The usage rate is random ranging from *0* to *1*. We follow the cost model presented in Section 3. And according to the Amazon's cost model, *$0.1 per CPU hour* for computation equals to *$1/600 per CPU min* and *$0.15 per gigabyte per month* equals to *$1/288000 per gigabyte per min* for storage.

As indicated in Table 1, the Table 1 shows how the nine strategies, which take the transfer cost into consideration, store the intermediate datasets in detail and the final results based on each strategy.

Table 1. The workflow's intermediate dataset storage status in the nine strategies

The dataset	0	1	2	3	4	5	The total cost
IDS_Greedy	S	S	S	S	S	C	20.07192901
Store all	S	S	S	S	S	S	20.87331789
Store none	S	C	C	C	C	C	20.24333329
Store top 20% often used datasets	S	C	C	S	C	C	20.14692897
Store top 40% often used datasets	S	S	C	S	C	C	20.11156440
Store top 60% often used datasets	S	S	S	S	C	C	20.08699845
Store top 20% generation datasets	S	C	C	C	C	S	20.97472219
Store top 40% generation datasets	S	C	C	S	C	S	20.92331790
Store top 60% generation datasets	S	C	C	S	S	S	20.89831784

For the *1th* dataset, although its size is quite high, comparing to high computing time and the low tolerance delay, it should be stored. However, in the strategy of "store top 20% generation datasets", it is chosen to be computed. For the *2th* dataset, it is not often used, but comparing to the high generation time and the small size, it should be stored. However, in the strategy of "store top 60% generation datasets", it is chosen to be deleted. For the *5th* dataset, although its generation cost is quite high, comparing to its usage frequency, it is not worth to store it. However, in the strategy of “store top 40% generation datasets”, it is chosen to be stored. Generally speaking, our strategy is the most appropriate strategy for IDSP.

As indicated in Fig. 2, we can draw the conclusions that (1) neither storing all the intermediate datasets nor storing top 60% generation intermediate datasets is a cost-effective method of intermediate datasets storage problem; (2) storing top 20% generation datasets is the worst method of IDSP; (3) the strategy of “store none” is in the middle band; (4) our strategy performs as the most cost effective method in comparison with the others.

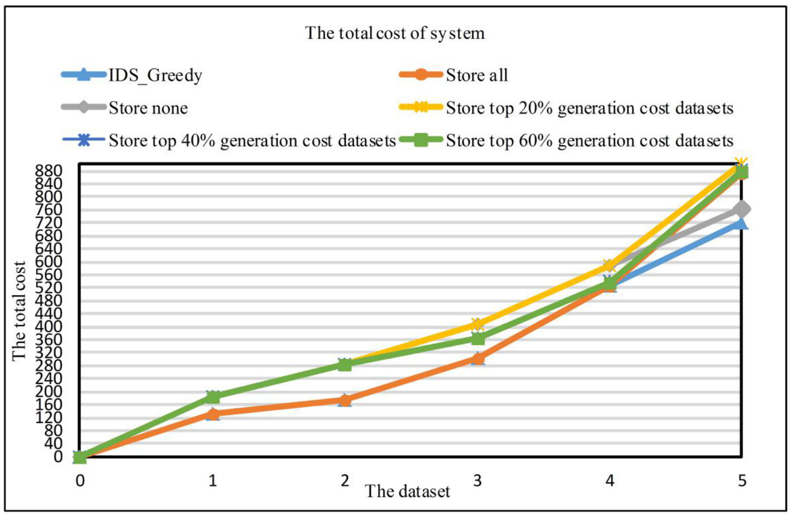

Fig. 2. Cost effectiveness evaluation of the “store high generation cost datasets”

As indicated in Fig. 3, we can draw the conclusions that (1) in the strategy of “store all”, sometimes the intermediate results are optimal, but the final result is the worst, which shows this strategy is feasible but not optimal; (2) the strategy of “store none” and the strategy of “store top 20% often used datasets” and the strategy of “store top 40% often used datasets” are in the middle band; (3) the cost of the strategy of “store none” is a most smooth line because in this strategy all the costs are computation costs of regenerating intermediate datasets; (4) the strategy of “store top 60% often used datasets” is closest to our strategy during all the process, but our strategy still performs the most cost effective strategy in comparison with it.

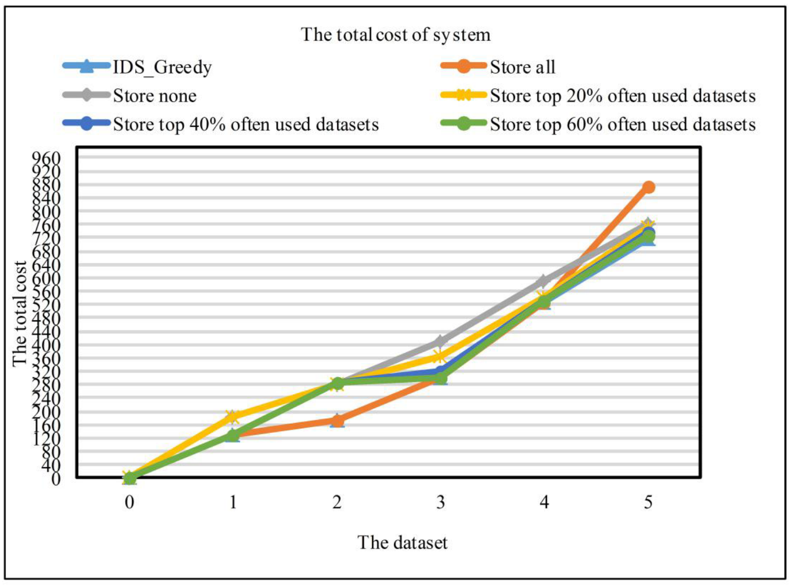

Fig. 3. Cost effectiveness evaluation of the “store often used datasets” strategy

5 Conclusions and future work

In this paper, we combine the three parameters into the cost model to design the new cost model, where they are usage frequencies and delaying tolerance and the transfer-cost. Then we apply the greedy approach to the new cost model, Experimental results demonstrated the cost-effective of the proposed approach in comparison with the other methods.

In our current work, the parameters of the cost model are the storage cost, computation cost, delaying tolerance, usage frequency and transfer-cost. However, in the real world, the extra cost might be caused by the limited location and the movement between the different data centres. To addition, in this paper, we focus the cost-effective results and did not emphasize much on the algorithm's efficiency, which is also a future work.

Acknowledgements

The author thanks reviewers for their constructive suggestions.

References

[1] 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, Lecture Notes in Computer Science 3165(2004) 11-20.

[2] B. Ludäscher, I. Altintas, C. Berkley, C. Berkley, D. Higgins, E. Jaeger, Scientific workflow management and the Kepler system, Concurrency & Computation Practice & Experience 18(10)(2005) 1039-1065.

[3] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, Taverna: a tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20(17)(2004) 3045-3054.

[4] E. Deelman, A. Chervenak, Data management challenges of data-intensive scientific workflows, in: Proc. IEEE International Symposium on CLUSTER Computing and the Grid. IEEE, 2008.

[5] A. Weiss, Computing in the clouds, Networker 11(4)(2007) 16-25.

[6] D. Yuan, L. Cui, W. Li, X. Liu, An algorithm for finding the minimum cost of storing and regenerating datasets in multiple clouds, IEEE Transactions on Cloud Computing 99(2015) 1-1.

[7] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Proc. Grid Computing Environments Workshop, IEEE, 2008.

[8] E. Deelman, G. Singh, M. Livny, B. Berriman, J.Good, The cost of doing science on the cloud: the Montage example, in: Proc. ACM/IEEE Conference on High PERFORMANCE Computing, SC, 2008.

[9] D. Yuan, Y. Yang, X. Liu, J. Chen, On-demand minimum cost benchmarking for intermediate data storage in scientific cloud workflow systems, Journal of Parallel & Distributed Computing 71(2)(2011) 316-332.

[10] L. Moreau, J. Freire, J. Futrelle, The open provenance model: an overview, in: Proc. the 2nd International Provenance and Annotation Workshop, 2008.

[11] M.I.M. Saad, K.A. Jalil, M. Manaf, Data provenance trusted model in cloud computing, in: Proc. International Conference on Research and Innovation in Information Systems, 2013.

[12] 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 & Computation Practice & Experience 24(9)(2012) 956-976.

[13] D. Yuan, Y. Yang, X. Liu, J. Chen, A cost-effective strategy for intermediate data storage in scientific cloud workflow systems, in: Proc. IEEE International Symposium on Parallel & Distributed Processing, IEEE, 2010.

[14] W. Xu, J. Chen, An approach to intermediate dataset storage in scientific workflows, Journal of Computational Information Systems 11(7)(2015) 2377-2384.

[15] J. Cheng, D. Zhu, B. Zhu, Improved algorithms for intermediate dataset storage in a cloud-based dataflow, Theoretical Computer Science 657(A)(2017) 48-53.

[16] G.B. Berriman, E. Deelman, G. Juve, M. Rynge, J.S. Vockler, The application of cloud computing to scientific workflows: a study of cost and performance, Philos Trans A Math Phys Eng Sci 371(1983)(2010) 20120066.

[17] F.M. Madani, S. Mokhtari, Virtual optical network embedding over elastic optical networks with set-up delay tolerance, in: Proc. Electrical Engineering, IEEE, 2015.

[18] A. Chervenak, E. Deelman, M. Livny, M.H. Su, Data placement for scientific applications in distributed environments, in: Proc. Ieee/acm International Conference on Grid Computing, IEEE, 2007.

[19] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschec, Giggle: a framework for constructing scalable replica location services, in: Supercomputing, in: Proc. ACM/IEEE 2002 Conference, IEEE, 2002.

[20] D. Yuan, Y. Yang, X. Liu, J. Chen, A data placement strategy in scientific cloud workflows, Future Generation Computer Systems 26(8)(2010) 1200-1214.