An Efficient Algorithm for Runtime Minimum Cost Data Storage and Regeneration for Business Process Management in Multiple Clouds

Junhua Zhang[1], Dong Yuan[2], Lizhen Cui[1(✉)], and Bing Bing Zhou[2]

[1] School of Computer Science and Technology, Shandong University, Jinan, China
z.jh@mail.sdu.edu.cn, clz@sdu.edu.cn

[2] School of Information Technology, The University of Sydney, Sydney, Australia
{dong.yuan,bing.zhou}@sydney.edu.au

Abstract. The proliferation of cloud computing provides flexible ways for users to utilize cloud resources to cope with data complex applications, such as Business Process Management (BPM) System. In the BPM system, users may have various usage manner of the system, such as upload, generate, process, transfer, store, share or access variety kinds of data, and these data may be complex and very large in size. Due to the pas-as-you-go pricing model of cloud computing, improper usage of cloud resources will incur high cost for users. Hence, for a typical BPM system usage, data could be regenerated, transferred and stored with multiple clouds, a data storage, transfer and regeneration strategy is needed to reduce the cost on resource usage. The current state-of-art algorithm can find a strategy that achieves minimum data storage, transfer and computation cost, however, this approach has very high computation complexity and is neither efficient nor practical to be applied at runtime. In this paper, by thoroughly investigating the trade-off problem of resources utilization, we propose a Provenance Candidates Elimination algorithm, which can efficiently find the minimum cost strategy for data storage, transfer and regeneration. Through comprehensive experimental evaluation, we demonstrate that our approach can calculate the minimum cost strategy in milliseconds, which outperforms the exiting algorithm by 2 to 4 magnitudes.

Keywords: Cloud computing · Business Process Management · Datasets storage and regeneration

1 Introduction

In recent years, the emergence and proliferation of cloud computing provides users on demand, redundant, inexpensive and scalable resources [1]. 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 for complex applications and data intensive applications [2], such as BPM Systems [3],

F. Daniel et al. (Eds.): BPM 2018 Workshops, LNBIP 342, pp. 348–360, 2019.
https://doi.org/10.1007/978-3-030-11641-5_28

which aim to be a "holistic management" approach to satisfy the needs of users in organization's business process and can generate variety of datasets of large amount. These generated data contain important intermediate or final results of computation, which may need to be stored for reuse and sharing [4]. The fast-growing cloud computing market along with more and more cloud service providers enable BPM system to have flexible ways to utilize multiple cloud services with different prices of computation, storage and bandwidth resources [5]. An efficient storage strategy which can cut the cost of multi-cloud-based data management in a pay-as-you-go fashion is in need for deploying applications in multi-cloud computing environment.

Furthermore, due to the dynamic property of usage of data, some data in the application could be more popular to the users at a certain time, some other data could be less popular, the usage frequency of data could vary from time to time, such as the data in BPM system [3], the efficiency of the data storage strategy that was efficient in a previous time could also degrades. To this end, an efficient algorithm that can generate the minimum cost storage strategy at runtime to keep low resource cost is very important for online data intensive applications in multi-cloud environment.

Finding the trade-off among computation, storage and bandwidth costs to achieve minimum total cost in multi-clouds is a complicated problem [6]. Different cloud service providers have different prices on their resources and datasets have different resource usage and generation dependencies. Even worse, the dynamic data usage frequencies demand that the storage strategy should be updated in time to avoid performance degradation. For this problem, our previous work [6] has proposed GT-CSB which can find the optimal storage strategy that has the minimum overall cost, however, this approach is impractical for runtime storage strategy due to high computation complexity. Therefore, it is necessary to design a highly efficient runtime algorithm that can find optimal storage strategy at runtime to adjust the data storage status in real time.

In this paper, by studying the intrinsic property of the minimum cost storage problem, we propose a dynamic programming algorithm which can reduce the searching space and find the optimal storage strategy in nearly linear time. We also propose optimizing strategies, which can help us calculate the (1) minimum regeneration cost in $O(m^2)$ and (2) the sum overall cost rate of dataset in $O(m)$ (m is the number of Cloud Service Providers). By conducting extensive experimental studies, we find that our algorithm has a very good performance and is scalable with large number of datasets and Cloud Service Providers.

The remainder of this paper is organized as follows: Sect. 2 discusses the related work. Section 3 analyses the problem and presents some preliminaries. Section 4 introduces the detail of PCE algorithm. Section 5 evaluate PCE algorithm. Section 6 concludes this paper.

2 Related Work

The resource management in clouds becomes a very important research topic, much work has been done about resource negotiation [7], replica placement [8] and multi-tenancy in clouds. Foster et al. [9] 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 [10].

Plenty of research has been done with regard to the tradeoff between computation and storage. The Nectar system [11] is designed for automatic management of data and computation in data centers, where obsolete data are deleted and regenerated whenever reused in order to improve resource utilization. In [12], authors firstly propose a cost-effective strategy based on the trade-off of computation and storage cost. In [13], the authors propose a dynamic on-the-fly minimum cost benchmarking approach by pre-storing calculated results with a specially designed data structure.

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 [14], Joe-Wong et al. investigate computation, storage and bandwidth resources allocation in order to achieve a trade-off between fairness and efficiency. In our prior work [15], we propose the T-CSB algorithm which can find a trade-off among Computation, Storage and Bandwidth costs (T-CSB). In our another prior work [6], we propose the GT-CSB algorithm, which can find a Generic best Trade-off among Computation, Storage and Bandwidth in clouds.

In this paper, to address above problem, we propose the PCE algorithm, which can efficiently find a Generic best Trade-off among Computation, Storage and Bandwidth in multiple clouds with a computation complexity of $O(n*|cand|*(m^2+log(|cand|)))$.

3 Preliminaries

In this Section, we first introduce some preliminaries and then the GT-CSB algorithm.

3.1 Preliminaries

In general, there are two types of data stored in clouds, *original data* and *generated data*, in this paper, we only consider *generated data*.

In this paper, we use *DDG* [16] (Data Dependency Graph) to represent datasets generation relationships. *DDG* [16] is a *DAG* which is based on data provenance in applications. Figure 1 depicts a simple DDG, where a node in the graph denotes a dataset. Edge denotes the generation relationship between datasets, i.e., d_4 and d_6 are needed for generation of d_7. If there exists a path from d_i to d_j in the *DDG*, we say d_i and d_j have a generation relationship, and $d_i(d_j)$ is the predecessor (successor) of $d_j(d_i)$, we denote it as $d_i \rightarrow d_j$, e.g., $d_1 \rightarrow d_4$, $d_5 \rightarrow d_7$.

In a commercial cloud computing environment, there are generally three basic types of resource cost in the cloud: computation cost, storage cost and bandwidth cost:

$$\textbf{\textit{Total Resource Cost}} = \textbf{\textit{Computation Cost}} + \textbf{\textit{Storage Cost}} + \textbf{\textit{Bandwidth Cost}}.$$

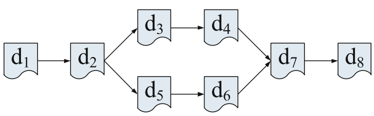


Fig. 1. A simple Data Dependency Graph (*DDG*)

Assumptions: We assume that the application be deployed with *m* Cloud Service Providers, denoted as $CSP = \{c_1, c_2 \ldots c_m\}$. Furthermore, we assume there are *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.

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 cost of computing d_i from its direct predecessors with cloud c_j;
$Y_{d_i}^{c_j}$ denotes the storage cost per time unit for storing dataset d_i with cloud c_j;
$Z_{d_i}^{c_k,c_j}$ denotes the cost of transferring dataset d_i from cloud service provider c_k to c_j.
v_{d_i} denote the usage frequency of d_i, which means how often d_i is accessed.

Definition 1: In a multi-cloud computing environment, in order to regenerate a deleted dataset, we need first to find its stored provenance dataset(s), then to choose a cloud service provider to regenerate it. We denote the minimum regeneration cost of dataset d_i as $minGenCost(d_i)$.

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 is:

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

The *Total Cost Rate* of a *DDG* is the sum *Cost Rate* of all the datasets: $TCR = \sum_{d_i \in DDG} CostR(d_i)$.

Definition 3: Storage strategy of a *DDG* is the storage status of all datasets in the *DDG*, i.e. whether dataset is stored, and which cloud the dataset is stored.

Definition 4: Minimum cost of a *DDG* is the minimum *Total Cost Rate* for storing and regenerating datasets in the *DDG*, which is denoted as $\mathrm{TCR_{min}} = \min\left(\sum_{d_i \in DDG} CostR(d_i)\right)$.

3.2 GT-CSB Algorithm

The GT-CSB algorithm proposed in our prior work [6] can find the best trade-off among computation, storage and bandwidth costs in multi-clouds. The core idea of GT-CSB is to convert a minimum cost storage problem to a shortest path problem over a Cost Transitive Graph (CTG) graph. In the CTG graph, for each dataset in *DDG*, there are *m* nodes each representing that the dataset is stored in the corresponding cloud, and two virtual vertexes, start vertex and end vertex, are used to represent the start point and end point of the shortest path problem. For any two vertexes belonging to different datasets, there is an edge between them. An edge signifies that the datasets between the edge are deleted while the end datasets of the edge are stored in the corresponding cloud. Each path from the start vertex to the end vertex in the CTG corresponds to a storage strategy of the datasets in the clouds. By sophistically setting the edge weight, which represents the sum *Cost Rate* of those datasets between the end nodes of the edge, we can get the minimum cost storage strategy by solving shortest path problem over the graph, the length of the shortest path corresponding to the minimum *Total Cost Rate* of datasets in *DDG*.

4 PCE Algorithm

In this section, we first detailed introduce our PCE algorithm and some optimizing strategies in Section; then we analyze the complexity of PCE algorithm.

4.1 Provenance Candidates Elimination (PCE) Algorithm

In this section, we will first elaborate the minimum cost dataset regeneration in Multiple Clouds Environment and baseline approach for optimal data storage strategy, and then introduce the detail of PCE Algorithm and optimizations.

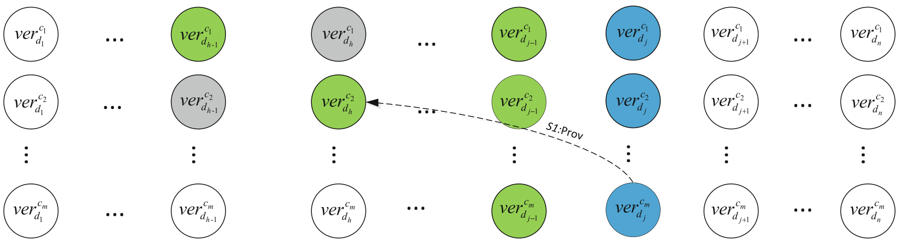


Fig. 2. *DDG* with multiple clouds

Dataset Regeneration with Multiple Clouds. We use *Prov*(*d*) to denote the provenance of dataset *d*, the provenance of *d* is the nearest stored predecessor(s) of *d* and is used to generate *d* when *d* is reused. The Minimum cost to regenerate a dataset is the minimum cost of generating the dataset from its provenance with multiple clouds, which includes the bandwidth cost for transferring datasets among the clouds and the computation cost for regenerating datasets from its predecessors.

Definition 5: We use $ver^{c_s}_{(d_j,c_k)d_i}$ to denote the minimum cost of generating d_i on cloud c_s from its provenance d_j which is stored in c_k, or simplify it as $ver^{c_s}_{d_i}$ in the context without ambiguity.

Based on the definition, if a provenance d_i is stored in cloud c_s, the minimum generation cost of dataset on cloud can be iteratively computed as:

$$\begin{cases} ver^{c_k}_{d_{i+1}} = Z^{c_s,c_k}_{d_i} + X^{c_k}_{d_{i+1}} \\ ver^{c_k}_{d_j} = \min_{h=1}^{m}\left\{ver^{C_h}_{d_{j-1}} + Z^{c_h,c_k}_{d_{j-1}}\right\} + X^{c_k}_{d_j} \end{cases} \quad (1)$$

where $d_j \in DDG \wedge d_{i+1} \rightarrow d_j \wedge \mathrm{Prov}(d_j) = d_i$, $c_k \in \{c_1, c_2, \ldots c_m\}$.

Based on Definition 5, the minimum regeneration cost of d_j with provenance d_i is:

$$minGenCost\left(d_j\right) = \min_{h=1}^{m}\left\{ver_{d_j}^{c_h}\right\} \tag{2}$$

Baseline Algorithm

Lemma 1. *In a linear DDG, if dataset $d_i \in DDG$ is stored in cloud, then the sum Cost Rate of d_i's successors (predecessors) is independent of the storage status of d_i's predecessors (successors).*

According to the definition and the iterative calculation of the minimum regeneration cost of a dataset in Eqs. (1) and (2), a deleted dataset is computed from its provenance, since d_i is stored in cloud, any of d_i's predecessor cannot be a provenance of d_i's successor, so the overall cost of d_i's successors is independent of the storage status of d_i's predecessors. The regeneration cost or storage cost of d_i's predecessors is also independent of d_i's successor. Hence, if a dataset, e.g. d_i, is stored in cloud, we can compute its predecessors' storage strategy and its successors' storage strategy independently.

Assume a dataset d_i is stored in cloud c_k, we use $d_i.preCost$ to represent the minimum total cost of d_i's predecessors, and a tuple (d_i, c_k) to represent that dataset d_i is stored in cloud c_k, and the storage strategy S of a DDG in multi-clouds is represented by a set of tuples $S = \{(d_i, c_k) | d_i \in DDG \wedge c_k \in CSP \wedge d_i \text{ is stored in } c_k\}$. The provenance, e.g. d_j, and the provenance stored place, e.g. c_k, of d_i is represented by a tuple $d_i.Prov = (d_j, c_k)$.

Algorithm 1. Baseline Algorithm

Input: a set of *CSPs* $\{cs_1 \ldots cs_m\}$ and a linear *DDG* $\{d_1 \ldots d_n\}$;
Output: a set of stored datasets and their storage clouds

1. Construct vitual dataset $d_0(d_{n+1})$ to be direct predecessor(successor) of $d_1(d_n)$;
2. Set d_0.size, d_0.computcost, d_0.preCost and d_1.preCost to 0;
3. **for each** dataset d_i in *DDG* from d_1 to d_{n+1} **do**
4.   Set d_i.preCost to Infinite;
5.   **for each** dataset d_j in *DDG* from d_0 to d_{i-1} **do**
6.   **for** each Cloud Service Provider c_k in *CSP* **do**
7.   Let d_j stored in c_k;
8.   **if**(d_j.preCost+ $\sum_{h=j+1}^{i-1} minGenCost(d_h) \times v_{d_h} + Y_{d_j}^{c_k}$)< d_i.preCost **then**
9. d_i.preCost= $\sum_{h=j+1}^{i-1} minGenCost(d_h) \times v_{d_h} + d_j$.preCost $+ Y_{d_j}^{c_k}$;
10. d_i.Prov=(d_j, c_k);
11.   Delete d_j from c_k;
12. // Collect the stored datasets and their stored clouds by backward traverse
13. $S = \emptyset$;
14. $d_s = d_{n+1}$;
15. **while** d_s.Prov.dataset$\neq d_0$ **do**
16. S=$S \cup \{d_s$.Prov$\}$;
17. $d_s = d_{n+1}$.Prov.dataset;

Baseline Algorithm starts by creating two virtual nodes d_0 and d_{n+1} as starts dataset and end dataset respectively (line 1), the two datasets have 0 size and 0 computation cost, they are created only for ease of illustration. For each dataset in *DDG* and d_{n+1}, e.g. d_i, Baseline-Algorithm computes its minimum *preCost* and *Prov* (line 5–11). After the iteration process on all datasets, d_{n+1}.*preCost* is the minimum total cost of all d_{n+1}'s predecessors and is also the minimum total cost of *DDG*, then the optimal storage strategy can be collected by a reverse traverse from d_{n+1} with *Prov* (line 13–17). When computing *preCost* and *Prov* of a dataset, e.g. d_i, *preCost* is first initialed as infinite, then Baseline-Algorithm iterates on all d_i's predecessors and all CSPs to determine d_i's provenance and the stored cloud service, e.g. d_i.Prov = (d_j, c_k), that can make d_i.*preCost* minimum (line 4–11).

In Baseline Algorithm, let *n* be the number of dataset and *m* be the number of CSPs, *minGenCost* can be compute in $O(m^2n)$. When deciding *Prov* of a dataset, Baseline-Algorithm have to iterate all its predecessors and all Cloud Service Providers, this procedure can be done in $O(m^3n^3)$, and there are *n* datasets, so the final time complexity of Baseline-Algorithm is $O(m^3n^4)$.

Provenance Candidates Elimination Strategy. Based on the definition of minimum regeneration cost in multiple clouds, we find that the more distant the $Prov(d_j)$ is from d_j, the higher the minimum regeneration cost of d_j will be.

Theorem 1: *In multi-cloud scenarios, without loss of generality, if exists an optimal storage strategy* $S1^*$ *for datasets* $\{d_1, d_2 \ldots d_j\}$*, i.e.,* $\sum_{i=1}^{j} CostR(d_i)$*, is minimum with* $S1^*$*, assuming the last stored dataset of* $S1^*$ *is* d_h *and is stored in cloud* c_r*, then the last stored dataset and its stored cloud of optimal storage strategy* $S2^*$ *for datasets* $\{d_1, d_2 \ldots d_j, d_{j+1}\}$ *cannot be* (d_k, c_i) *with* $ver^{c_s}_{(c_r,d_h)d_{j+1}} < ver^{c_s}_{(c_i,d_k)d_{j+1}}$ *for all* $c_s \in CSP$*.*

Proof: Assuming the last stored dataset of $S2^*$ is (d_k, c_i) with $ver^{c_s}_{(c_r,d_h)d_{j+1}} < ver^{c_s}_{(c_i,d_k)d_{j+1}}$ for all $c_s \in CSP$. We can construct a strategy *S3* for $\{d_1, d_2, \ldots, d_{j+1}\}$ with same storage strategy of $S1^*$ for $\{d_1, d_2, \ldots, d_j\}$ and d_{j+1} is deleted with lower sum *Cost Rate* than $S2^*$. Since $\sum_{i=1}^{j} CostR_{S1^*}(d_i) < \sum_{i=1}^{j} CostR_{S2^*}(d_i)$ and $ver^{c_s}_{(c_2,d_h)d_{j+1}} < ver^{c_s}_{(c_i,d_k)d_{j+1}}$ for all $c_s \in$ CSP, $CostR_{S3}(d_{j+1}) = v_{d_{j+1}} \times \min_{c_s \in CSP} ver^{c_s}_{(c_2,d_h)d_{j+1}} < \mathrm{Cost}R_{S2}(d_{j+1}) + v_{d_{j+1}} \times \min_{c_s \in CSP} ver^{c_s}_{(c_2,d_k)d_{j+1}}$, hence $\sum_{i=1}^{j+1} CostR_{S3}(d_i) = \sum_{i=1}^{j} \mathrm{Cost}R_{S1}(d_i) + \mathrm{Cost}R_{S3}(d_{j+1}) < \sum_{i=1}^{j+1} CostR_{S2}(d_i) = \sum_{i=1}^{j} CostR_{S2}(d_i) + CostR_{S2}(d_{j+1})$, which contradicts the premise. Theorem 1 holds.

According Theorem 1, we propose following Provenance Candidates Elimination Rules (*PCERs*).

Consider the Baseline-Algorithm, assume the provenance of a dataset d_i is d_i.*Prov* = (d_j, c_k), for d_i's successors, i.e., d_k, the initial provenance candidates set of d_k is d_k.*cand* = $\{(d_h, c_l) | \, d_h \rightarrow d_k \wedge d_h \in DDG \wedge c_l \in CSP\}$, we can use the following rules to pruning the candidates set:

1. For $(d_h, c_l) \in d_k$.*cand*, where $d_h \rightarrow d_i$, if $ver^{c_s}_{(d_h,c_l)d_i} > ver^{c_s}_{(d_k,c_j)d_i}$ for all $c_s \in CSP$, then (d_h, c_l) can be eliminated from d_k.*cand*.

2. For (d_h, c_l) $d_k.cand$, where $d_h \rightarrow d_i$, if exists $(d_{h'}, c_{l'})$ $d_k.cand$, $d_{h'} \rightarrow d_i$, that $\left(d_h.preCost + \sum_{p=h+1}^{i}\left(\min_{c_s \in CSP}\left(ver^{c_s}_{(d_h,c_l)d_p}\right) \times v_{d_p}\right) + Y^{c_l}_{d_h}\right) > (d_{h'}.\text{preCost} + \sum_{p=h+1}^{i}\left(\min_{c_s \in CSP}\left(ver^{c_s}_{(d_{h'},c_{l'})d_p}\right) \times v_{d_p}\right) + Y^{c_{l'}}_{d_{h'}})$ and $ver^{c_s}_{(d_h,c_l)d_i} > ver^{c_s}_{(d_{h'},c_{l'})d_i}$, then (d_h, c_l) can be eliminated from $d_k.cand$.

To better illustrate the PCE Algorithm, we first introduce some new data structures:

- *cand* is the candidates set to record the possible provenances of the datasets. In the algorithm, maintaining one *cand* is sufficient for all datasets, because, for example, the reduction on a dataset's provenance candidates d_i.cand also applies on d_i's successors.
- $(d_j, c_k).MGC$ is an array where $(d_j, c_k).MGC[c_s]$ is the value of $ver^{c_s}_{(d_j,c_k)d_{i-1}}$ when d_j is stored in cloud c_k and d_{i-1} is generated on cloud c_s.
- $(d_j, c_k).sucCost$ is similar to $d_j.preCost$, it is the sum $CostR$ of datasets from d_{j+1} to d_{i-1} : $(d_j, c_k).sucCost = \sum_{h=j+1}^{i-1} minGenCost(d_h) \times v_{d_h}$.

Algorithm 2. PCE Algorithm	
Input:	a set of Cloud Service Providers *CSP* $\{cs_1, cs_2 \ldots cs_m\}$ a linear *DDG* $\{d_1, d_2 \ldots d_n\}$;
Output:	a set of stored datasets and their storage clouds

Construct vitual dataset $d_0(d_{n+1})$ to be direct predecessor(successor) of $d_1(d_n)$;
Set d_0.size, d_0.computcost, d_0.preCost, (d_0,c_0).sucCost, d_1.preCost to be zero;
(d_0, c_0).MGC[c_s]=Infinit for all c_s in *CSP* except 0 when c_0==c_s;
cand=$\{(d_0, c_0)\}$;
for each dataset d_i in *DDG* from d_1 to d_{n+1} **do**
  d_i.preCost=Infinite;
  for each (d_j, c_k) in cand **do**
    if (d_j.preCost+(d_j, c_k).sucCost+$Y^{c_k}_{d_j}$)<d_i.preCost **then**
      d_i.preCost= d_j.preCost+(d_j, c_k).sucCost+$Y^{c_k}_{d_j}$;
      d_i.Prov=(d_j, c_k);
  //Incremental Update
  for each (d_j, c_k) in cand **do**
    Swap (d_j, c_k).MGC with (d_j, c_k).MGCO;
    for each c_s in *CSPs* **do**
      (d_j, c_k).MGC[c_s]= $\min_{h=1}^{m}\{(d_j,c_k).\text{MGCO}[c_h]+Z^{c_h,c_s}_{d_{i-1}}\}+X^{c_s}_{d_i}$;
    (d_j, c_k).sucCost=(d_j, c_k).sucCost+ $\min_{h=1}^{m}\{(d_j,c_k).\text{MGC}[c_h]\}\times v_{d_i}$;
  Performing Provenance Candidates Elimination Rule 1 and 2 on *cand*;
  //Adding new candidates for d_{i+1}
  for each c_k in *CSP* **do**
    (d_i, c_k). MGC[c_s]=Infinit for all c_s in *CSP* except 0 hen c_k==c_s;
    (d_i, c_k).sucCost=0;
    cand=cand ∪ $\{(d_i, c_k)\}$;
Collect the stored datasets and their stored clouds by backward traverse

In PCE algorithm, the *cand* is first initialized as $\{(d_0, c_0)\}$ (line 4). For each d_i in *DDG*, d_i.Prov and d_i.preCost computed in line 6–10, after updating *MGC* and sucCost of all the candidates (line 12–16), the *PCERs* are performed on *cand* (line 17). At last in line 19–22, the new candidates, i.e., (d_i, c_k) for all c_k in *CSP*, are initialized and added to *cand*.

For example, in Fig. 2, the provenance of d_j is (d_h, c_2), the *cand* now is $\{(d_{h-1}, c_1), (d_{h-1}, c_2), (d_h, c_1), (d_h, c_2), (d_{j-1}, c_1), (d_{j-1}, c_2), (d_{h-1}, c_m)\ldots\}$ marked with grey and green circles. After performing the elimination rules, (d_{h-1}, c_2) and (d_h, c_1) marked with grey circles are deleted from *cand*. Then before searching Prov of d_{j+1}, (d_j, c_1), (d_j, c_2) … (d_j, c_m) marked with blue circles are added to *cand*.

Incremental Minimum Regeneration Cost and Sum Successors' Cost. For the computation of $\sum_{h=j+1}^{i-1} minGenCost(d_h) \times v_{d_h}$, we propose incremental computation for it, it contains two parts: the incremental computation of $minGenCost(d_h)$ and $\sum_{h=j+1}^{i-1} minGenCost(d_h) \times v_{d_h}$, as was illustrated in line 12–16 of PCE algorithm.

First, for the computation of $minGenCost(d_h)$, we use a data structure *MGC*, introduced before, to store the minimum regeneration cost of successors of datasets, e.g. (d_j, c_k).*MGC* stores the minimum regeneration cost of successors of d_j. In the each round, *MGC* is updated accordingly (line 15).

Second, for the computation of $\sum_{h=j+1}^{i-1} minGenCost(d_h) \times v_{d_h}$, similar to the incremental computation of $minGenCost(d_h)$, we use *sucCost*, introduced before, to store the sum cost rate of successors of a datasets, e.g., d_j. In each round, *sucCost* is updated accordingly (line 16).

4.2 Analyses

In PCE Algorithm, let n be the number of datasets, m be the number of Cloud Service Providers and $|cand|$ be the average size of *cand*, searching of *Prov* (line 6–10) can be done in $O(|cand|)$, incremental update(line 12–16) can be done in $O(|cand|*m^2)$, elimination rules (line17) in $O(|cand|*m+|cand|*log(|cand|))$, adding new candidates (line 26–29) can be done in $O(m^2)$, so the overall time complexity of the Algorithm is $O(n*|cand|*(m^2+log(|cand|)))$. For the size of *cand*, it mainly depends on the computation cost rate and storage cost rate of datasets and is independent of the number of datasets n. Our experimental results in Sect. 5.2 (Fig. 4(b)) also demonstrate the independence of the size of *cand* and the number of dataset n.

5 Experiments

Our experiment is conducted on Desktop PC with Intel(R) Core(TM) i5-4200M CPU, RAM 8 GB. The algorithm is implemented in the Java and is run on Windows.

In real world applications, generated datasets may vary dramatically in terms of size, generation time, usage frequency and the structure of *DDG*. Hence, we randomly generate *DDGs* with different number of datasets, each with a random size from 1 GB to 100 GB. The computation time of dataset is also random, from 10 h to 100 h.

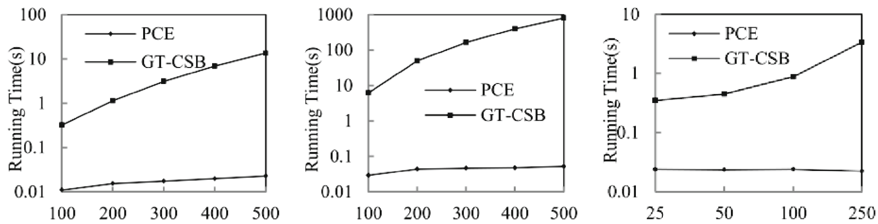
The usage frequency is again random, from once per month to once per year. This setting is based on the scenarios of applications of scientific workflow [16] and BPM system [3].

Table 1. The pricing models of 10 cloud services providers

Cloud Service ID	0	1	2	3	4	5	6	7	8	9
Compute cost rate ($/hour)	0.11	0.12	0.15	0.09	0.13	0.15	0.12	0.13	0.12	0.16
Storage cost rate ($/GB*month)	0.1	0.06	0.05	0.08	0.07	0.07	0.06	0.09	0.05	0.04
Transfer cost rate for outbound ($/GB)	0.01	0.03	0.15	0.05	0.06	0.03	0.07	0.02	0.06	0.08

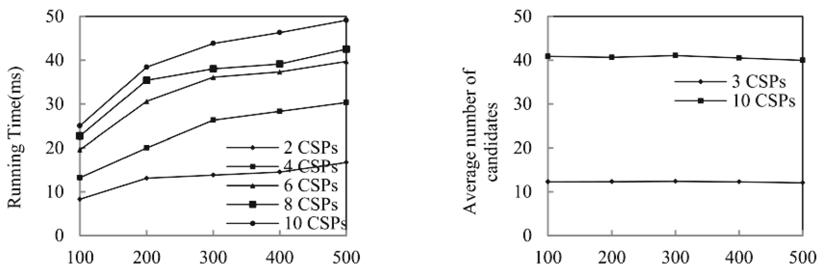
In addition, we randomly generate 10 cloud service providers with different compute, storage and out-bandwidth price (see Table 1)[1].

Our prior work [6] has thoroughly investigated the minimum cost strategy, the algorithm in this paper calculates the same minimum cost strategy as GT-CSB, the effectiveness of PCE algorithm will not be evaluated here.



(a) Varying #Datasets (3 *CSPs*) (b) Varying #Datasets (10 *CSPs*) (c) Varying Block Size(3 *CSPs*)

Fig. 3. Comparison of performance with varying settings



(a) Running time with varying #Datasets (b) Avg. candidates size with varying #Datasets

Fig. 4. Evaluation with varying settings

[1] The prices are set based on popular cloud service provider's pricing model, e.g., Amazon Web Services' prices are: $0.10 per instance-hour for the computation resources, $0.10 per GB-month for the storage resource and $0.09 per GB bandwidth resources for data downloaded from Amazon via the Internet. https://aws.amazon.com 2018.

5.1 Comparison with Existing Algorithms

We first compare the performance of our strategy with GT-CSB. In this experiment, we use 5 randomly generated DDGs with 100 to 500 datasets and 3 cloud service providers with the pricing models listed in Table 1.

The experiment result shown in Fig. 3(a) and (b) demonstrates our strategy can always finished within 1 s, while the running time of GT-CSB increases fast with the increase of number of datasets.

In the next experiment, based on the philosophy of our prior work [17], we devise a method which can derive localized minimum cost instead of a global one. The method is dividing the DDG into several blocks of the same size, and using the algorithm to find local optimal storage strategy for each block. We use a DDG with 500 datasets and divide it into blocks with different block size. Figure 3(c) demonstrate the speed up of GT-CSB algorithm with small block size, however, it is still not as efficient as PCE algorithm.

5.2 Evaluation of PCE with Varying Settings

Then we evaluate the efficiency of our strategies with varying number of cloud service providers.

We use the same datasets as above experiment, but gradually increase the number of cloud service providers. All cloud service providers are summarized in Table 1. As can be seen in Fig. 4(a), the run time of our algorithm increase slowly when the number of datasets or the number of cloud service providers increases. Compared with existing work, with the pruning effect of provenance candidates elimination and incremental computation, our algorithm can complete in near linear time in terms of number of datasets, hence, even if we use 10 cloud service providers and the 500 datasets, we can get the result in approximate 50 ms.

We demonstrate the effect of provenance elimination strategy by studying the average number of candidates with varying number of datasets (100–500). The number of candidates indicates how many times we should check before we could get the optimal provenance of a dataset, which is a key factor to the algorithm efficiency. In this experiment, we summarized the average number of candidates with 3 and 10 cloud service providers separately, as show in Fig. 4(b). With the varying number of datasets, the average number of candidate remains almost constant, which demonstrate that the number of candidates is independent of the number of datasets.

6 Conclusions and Future Work

In this paper, we proposed a provenance elimination strategy which can identify a small set of possible optimal provenance and reduce the search space. Besides, we propose incremental computations which speed up the algorithm a lot. The experimental results show that the running time of our algorithm is significantly reduced compared to that of the GT-CSB algorithm and our algorithm also scales well even the number of dataset is very large.

In our current work, we only consider the datasets with linear *DDG*. However, in the real world, dependencies between datasets can be very complex; they may contain blocks, sub-blocks and crossed-blocks, the data storage strategy can be very tough to obtain. 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) intensive applications, etc. In the future, we will consider complex *DDG* and incorporate more complex pricing models in our datasets storage and regeneration cost model.

Acknowledgment. The research work was supported by the National Key R&D Program (2017YFB1400102, 2016YFB1000602), NSFC (61572295), SDNSFC (No. ZR2017ZB0420), and Shandong Major scientific and technological innovation projects (2018YFJH0506).

References

1. Zhang, Q., Zhani, M.F., Boutaba, R., Hellerstein, J.L.: Dynamic heterogeneity-aware resource provisioning in the cloud. IEEE Trans. Cloud Comput. **2**(1), 14–28 (2014)
2. Szalay, A., Gray, J.: 2020 computing: science in an exponential world. Nature **440**(7083), 413–414 (2006)
3. Weske, M.: Business process management architectures. Business Process Management, pp. 333–371. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28616-2_7
4. Burton, A., Treloar, A.: Publish my data: a composition of services from ANDS and ARCS. In: Fifth IEEE International Conference on e-Science, pp. 164–170. IEEE (2009)
5. Agarwala, S., Jadav, D., Bathen, L.A.: iCostale: adaptive cost optimization for storage clouds. In: 4th International Conference on Cloud Computing, pp. 436–443. IEEE (2011)
6. Yuan, D., Cui, L., Li, W., Liu, X., Yang, Y.: An algorithm for finding the minimum cost of storing and regenerating datasets in multiple clouds. IEEE Trans. Cloud Comput. **6**, 519–531 (2015)
7. Deng, K., Song, J., Ren, K., Yuan, D., Chen, J.: 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, pp. 34–41. IEEE Computer Society (2011)
8. Li, W., Yang, Y., Chen, J., Yuan, D.: 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), pp. 564–571. IEEE Computer Society (2012)
9. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: Chimera: a virtual data system for representing, querying, and automating data derivation. In: Proceedings of 14th International Conference on Scientific and Statistical Database Management, pp. 37–46. IEEE (2002)
10. Muniswamy-Reddy, K.-K., Macko, P., Seltzer, M.I.: Provenance for the cloud, pp. 14–15 (2010)
11. Gunda, P.K., Ravindranath, L., Thekkath, C.A., Yu, Y., Zhuang, L.: Nectar: automatic management of data and computation in datacenters. In: OSDI, pp. 1–8 (2010)
12. Yuan, D., Yang, Y., Liu, X., Chen, J.: A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In: Parallel & Distributed Processing (IPDPS), pp. 1–12. IEEE (2010)
13. Yuan, D., Liu, X., Yang, Y.: Dynamic on-the-fly minimum cost benchmarking for storing generated scientific datasets in the cloud. IEEE Trans. Comput. **64**(10), 2781–2795 (2015)

14. Joe-Wong, C., Sen, S., Lan, T., Chiang, M.: Multiresource allocation: fairness-efficiency tradeoffs in a unifying framework. IEEE/ACM Trans. Netw. (TON) **21**(6), 1785–1798 (2013)
15. Yuan, D., et al.: An algorithm for cost-effectively storing scientific datasets with multiple service providers in the cloud. In: 2013 IEEE 9th International Conference on eScience (eScience), pp. 285–292 (2013)
16. Yuan, D., Yang, Y., Liu, X., Chen, J.: On-demand minimum cost benchmarking for intermediate dataset storage in scientific cloud workflow systems. J. Parallel Distrib. Comput. **71**(2), 316–332 (2011)
17. Yuan, D., et al.: A highly practical approach toward achieving minimum data sets storage cost in the cloud. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1234–1244 (2013)