A New Algorithm for Intermediate Dataset Storage in a Cloud-Based Dataflow

Jie Cheng[1], Daming Zhu[2](✉), and Binhai Zhu[3]

[1] School of Mechanical, Electrical and Information Engineering, Shandong University, Weihai, China

[2] School of Computer Science and Technology, Shandong University, Jinan, China
{chjie,dmzhu}@sdu.edu.cn

[3] Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA
bhz@cs.montana.edu

Abstract. Running a dataflow in a cloud environment usually generates many useful intermediate datasets. A strategy for running a dataflow is to decide which datasets should be stored, while the rest of them are regenerated. The intermediate dataset storage (IDS) problem asks to find a strategy for running a dataflow, such that the total cost is minimized. The current best algorithm for linear-structure IDS takes $O(n^4)$ time, where "linear-structure" means that the structure of the datasets in the dataflow is a pipeline. In this paper, we present a new algorithm for this problem, and improve the time complexity to $O(n^3)$, where n is the number of datasets in the pipeline.

1 Introduction

A cloud-based dataflow is a data-driven workflow deployed in a cloud computing environment. In a cloud-based dataflow, there are usually a large number of datasets, including initial dataset, output dataset and a large volume of intermediate datasets generated during the execution. The intermediate datasets often contain valuable intermediate results, thus would be frequently traced back for re-analyzing or re-using [1]. Since the dataflow systems are executed in a cloud computing environment, all the resources used need to be paid for. As indicated in [2], storing all of the intermediate datasets may induce a high storage cost, while if all the intermediate datasets are deleted and regenerated when needed, the computation cost of the system may also be very high. Hence, an optimal strategy is needed to store some datasets and regenerate the rest of them when needed so as to minimize the total cost of the whole workflow system [3,4], which is called the intermediate dataset storage (IDS) problem.

In a cloud dataflow system, when a deleted dataset needs to be regenerated, the computation cost will involve not only itself but its direct predecessors, if these predecessors are also deleted. Hence, the computation cost of a sequence of deleted datasets needs to be accumulated, which leads to the IDS problem. In [2], Yuan *et al.* presented the background of the IDS problem in scientific

J. Wang and C. Yap (Eds.): FAW 2015, LNCS 9130, pp. 33–44, 2015.
DOI: 10.1007/978-3-319-19647-3_4

workflows and proposed an intermediate data dependency graph (IDG). Based on IDG, they presented two algorithms as the minimum cost benchmark of the IDS problem, a linear CTT-SP algorithm for linear workflow which takes $O(n^4)$ time, and a general CTT-SP algorithm for parallel structure workflow which takes $O(n^9)$ time. Besides [2], there have been some related research. Zohrevandi and Bazzi [5] presented a branch-and-bound algorithm for the common intermediate dataset storage between two scientific workflows, which is related to the IDS problem. Adams *et al.* [3] proposed a model balancing the computation cost and the storage cost. The approach proposed by Han *et al.* [6] is to support automatic intermediate data reusing for large-scale cloud dataflow based on Perti-nets. As far as we know, the current best exact algorithm for the IDS problem is the one proposed by Yuan *et al.* in [2].

This paper focuses on the IDS problem for linear-structure cloud dataflow systems. We present a binary tree model that is called S-C tree for the IDS problem. In an S-C tree, a vertex represents a choice of a dataset, which could be storage or computation, and the price of a vertex represents the generation cost for the choice. Based on the S-C tree model, the optimal solution to the IDS problem can be converted to searching for an optimal full path in the S-C tree with the minimum path cost. To reduce the searching space, we propose a group of pruning strategies, by which, more than $\frac{k-1}{2k}$ of the branches will be pruned off at each level k. Therefore, with the increasing of the searching level, the searching space grows linearly. Using these pruning strategies, we present an exact algorithm for the linear-structure IDS problem and prove that the algorithm takes $O(n^3)$ time.

The rest of the paper is organized as follows. Section 2 introduces the IDS problem and some related concepts are defined there. The S-C tree model of the IDS problem, including the proof of some theorems, are presented in Sect. 3. Section 4 describes the algorithm based on the em S-C tree and the corresponding analysis. Section 5 concludes the paper.

2 The IDS Problem

In this section, we first introduce some related concepts, and then give the definition of the IDS problem.

Definition 1. *A linear-structure cloud dataflow F can be expressed as $F = (DS, TS)$, where,*

- $DS = \{d_0, d_1, \cdots, d_n\}$ *is a set of datasets, where n is the number of intermediate datasets. d_0 denotes the initial dataset, and d_n is the output dataset of F. For each $d_i, 0 < i < n$, d_{i-1} is the direct predecessor of d_i, and d_{i+1} is the direct successor of d_i;*
- $DT = \{t_1, t_2, \cdots, t_n\}$ *is a set of tasks, where $t_i, 0 \leq i \leq n$, is a logical computation unit executed using d_{i-1} as the input and outputs the dataset d_i. Given a dataset $d_i, 0 \leq i \leq n$, t_i is called the execution task of d_i.*

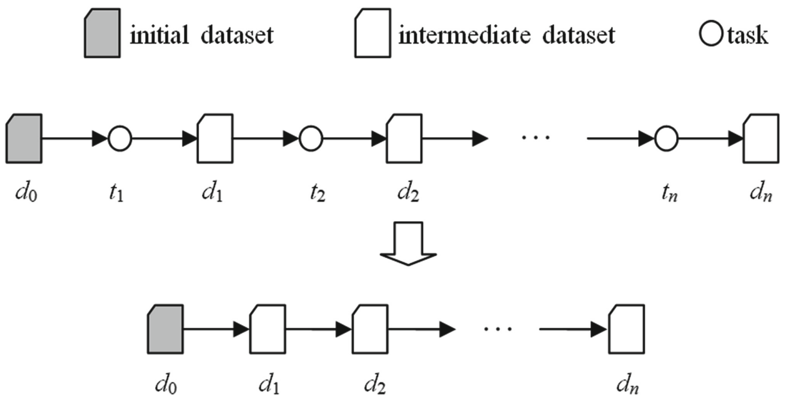


Fig. 1. The exemplar graph of a linear dataflow.

For simplicity, the linear-structure cloud dataflow is simply called *dataflow* throughout the rest of the paper. Since this paper focuses on datasets, a dataflow can also be simplified as a sequence of datasets, denoted as $F = \{d_0, d_1, \cdots, d_n\}$, as shown in Fig. 1.

As mentioned in [1], there are two basic types of resources in cloud computing: storage and computation. Normally, the service price of a cloud platform is proportional to the size of storage resource and also to the instance-hour for computation resource.

Given a dataflow F, we say that a dataset d is a storage dataset if d is selected to be stored; otherwise, it is a computation dataset. Thus F can be separated into two subsets, denoted as $F = S \cup C$, where S is the set of storage datasets and C is the set of computation datasets. We assume that the initial dataset d_0 is a storage dataset.

In a dataflow $F = \{d_0, d_1, \cdots, d_n\}$, there are two ways to generate an intermediate dataset $d_i, (0 < i \leq n)$, storage and computation. That is, if $d_i \in S$, it is available when needed; otherwise, it is deleted and has to be regenerated by computation. Therefore, there have two kinds of costs related to d_i, which are storage cost $x(d_i)$ if $d_i \in S$ and computation cost $y(d_i)$ if $d_i \in C$. In general, $x(d_i)$ is proportional to the size of d_i, and $y(d_i)$ is proportional to the running time of the execution task t_i.

Definition 2. *Given a dataset $d_j (j > 0)$, we say that the dataset d_i is the* **storage-prior** *of $d_j, 0 \leq i < j$, denoted as $d_i \mapsto d_j$, if $d_i \in S$ and for any $i < k < j, d_k \in C$. That is, $d_i \mapsto d_j$ means that d_i is the nearest predecessor storage dataset of d_j, as shown in Fig. 2.*

As indicated in [2], when we want to regenerate a computation dataset d_j, we have to find its direct predecessor d_{j+1} which may also be deleted, so we have to further trace the nearest stored predecessor, the storage-prior dataset of d_j.

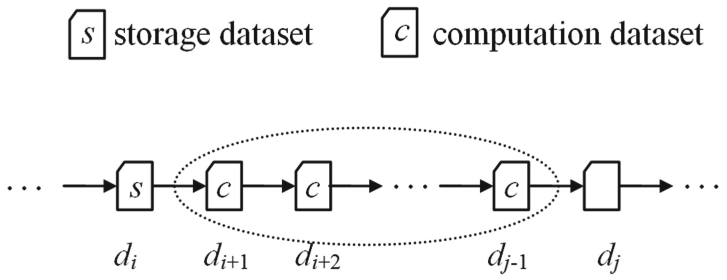


Fig. 2. An exemplar graph of $d_i \mapsto d_j$

Hence, for any intermediate dataset d_j, its generation cost is defined as:

$$G\_cost(d_j) = \begin{cases} x(d_j), & \text{if } (d_j \in S) \\ \sum_{k=i+1}^{j} y(d_k), & \text{if } (d_j \in C) \wedge (d_i \mapsto d_j) \end{cases} \tag{2-1}$$

Based on the above concepts, the IDS problem is defined as follows.

Input: given a dataflow $F = \{d_0, d_1, \cdots, d_n\}$, for each intermediate dataset d_i, its storage cost $x(d_i)$ and its computation cost $y(d_i)$.
Output: the set of storage dataset S and the set of computation dataset C.
Objective: the total cost of F, $\sum_{k=1}^{n} G\_cost(d_k)$, is minimized.

3 Binary Tree Model for the IDS Problem

The objective of the IDS problem is to find an optimal mapping between the intermediate datasets and the set C or S. Since a dataset has only two choices, we apply a binary tree as the problem model.

3.1 S-C Tree Model

Definition 3. *An* **S-C tree** *of a given dataflow $F = \{d_0, d_1, \cdots, d_n\}$, denoted as $Tree^F$, is full binary tree with $n+1$ levels, in which:*

(1) The root represents the initial dataset d_0.
(2) The set of nodes at the i^{th} level in $Tree^F$,$0 \le i \le n$, denoted as $N \mid_{Tree^F}^{i}$, is mapped to the dataset d_i.
(3) Any node $\tau \in N \mid_{Tree^F}^{i}, 0 \le i < n$, its left child $left(\tau)$ and right child $right(\tau)$ represent that the dataset d_{i+1} is selected to be stored and deleted respectively.

Figure 3 shows a 5-level S-C tree. According to Definition 3, the set of nodes in $Tree^F$ can be separated into $S = \{s_0, s_1, \cdots, s_{2^n-1}\}$ and $C = \{c_0, c_1, \cdots, c_{2^n-1}\}$, which are mapped respectively to the set of storage datasets and set of computation datasets in F. We can see that set S is composed of the root s_0 and all the left-child nodes, and set C consists of all the right-child nodes. The nodes of set S and C are also simply called S-nodes and C-nodes respectively.

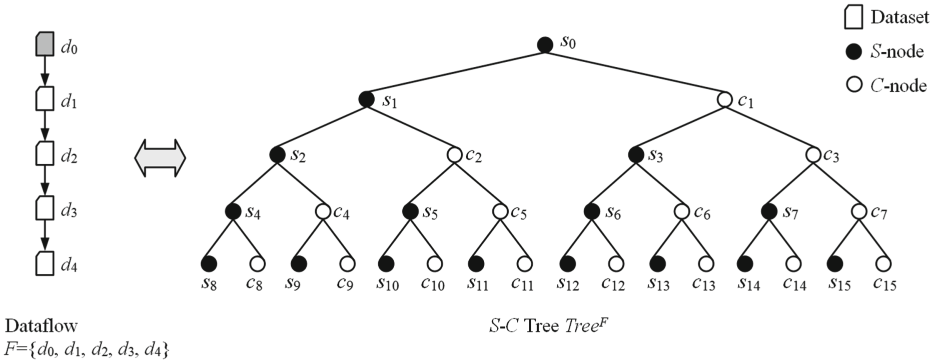

Dataset
S-node
C-node
d_0
d_1
d_2
d_3
d_4
Dataflow
$F=\{d_0, d_1, d_2, d_3, d_4\}$
s_0
s_1
c_1
s_2
c_2
s_3
c_3
s_4
c_4
s_5
c_5
s_6
c_6
s_7
c_7
s_8 c_8 s_9 c_9 s_{10} c_{10} s_{11} c_{11} s_{12} c_{12} s_{13} c_{13} s_{14} c_{14} s_{15} c_{15}
S-C Tree $Tree^F$

Fig. 3. An exemplar graph of 5-level S-C tree.

Definition 4. *Given an* S-C *tree* $Tree^F$*, the* **S-prior** *of a* C *node* τ*, denoted as* $\widehat{\tau}$*, is the nearest* S*-ancestor of* τ*. We use* $Path_{\tau}^{\mapsto}$ *to denote the ordered set of nodes of the path that is from* $\widehat{\tau}$ *to* τ*.*

For example, in Fig. 3, $\widehat{c}_4 = s_2, \widehat{c}_5 = s_1$. $Path_{c_5}^{\mapsto} = \{s_1, c_2, c_5\}, Path_{c_7}^{\mapsto} = \{s_0, c_1, c_3, c_7\}$. As we can see, in $Path_{\tau}^{\mapsto}$, only the first node $\widehat{\tau}$ is an S-node, others are C-nodes.

Definition 5. *Given a dataflow* $F = \{d_0, d_1, \cdots, d_n\}$ *and its* S-C *tree* $Tree^F$*, assume that* $\tau \in N\,|^{i}_{Tree^F}, 0 \le i < n$*, the* **weight** *and* **cost** *of* τ *are defined as follows.*

$$w(\tau) = \begin{cases} 0, & \text{if } \tau \in S \\ y(d_i), & \text{if } \tau \in C \end{cases} \tag{3-1}$$

$$cost(\tau) = \begin{cases} x(d_i), & \text{if } \tau \in S \\ \sum_{\alpha \in Path_{\tau}^{\mapsto}} w(\alpha), & \text{if } \tau \in C \end{cases} \tag{3-2}$$

We assume that $cost(s_0) = 0$.

Definition 6. *Given an* S-C *tree* T *with* $n+1$ *levels, a* **path** $P = \{p_a, p_{a+1}, p_{a+2}, \cdots, p_b\}, 0 \le a \le b \le n$*, is an ordered set of nodes from* p_a *to* p_b*, in which* $p_i \in N\,|^{i}_{T}$*,* $a \le i < b$*, and* p_i *is the parent node of* p_{i+1}*. The cost of path* P *is defined as:* $Cost(P) = \sum_{i=a}^{b} cost(p_i)$*.*

Definition 7. *Given an* S-C *tree* T *with* $n+1$ *levels, a* k**- path** $P^k = \{p_0, p_1, p_2, \cdots, p_k\}$*,* $0 \le k \le n$*, is a path in which the first node is the root of* T*. An* n*-path is also called a* **full path***. A full path* Λ *is called an* **optimal full path** *if and only if for any* **full path** Λ'*,* $Cost(\Lambda) \le Cost(\Lambda')$*.*

For example, in Fig. 3, $\{s_0, s_1, c_2, c_5\}$ is a 3-path, and $\{s_0, s_1, c_2, c_5, c_{11}\}$ is a full path.

Given a k-path $P^k = \{p_0, p_1, p_2, \cdots, p_k\}$ and a path $P = \{p_k, p_{k+1}, p_{k+2}, \ldots, p_{k+i}\}$, we use $link(P^k, P)$ to denote the $(k+i)$-path: $\{p_0, p_1, p_2, \cdots, p_k, p_{k+1}, p_{k+2}, \cdots, p_{k+i}\}$. In addition, if τ is a child node of p_k, we also use $link(P^k, \tau)$ to denoted the $(k+1)$-path: $\{p_0, p_1, p_2, \ldots, p_k, \tau\}$.

Definition 8. *Given an* S-C *tree T with $n+1$ levels, let $\tau \in N\,|_T^k, 0 \leq k \leq n$, the sub-tree which is rooted at τ is called a k-***subtree***. A k-subtree δ is called* **an optimal k-subtree** *if and only if δ contains an optimal full path from the k^{th} level to the n^{th} level.*

For example, an *S-C* tree itself is an optimal 0-subtree. Assume that $\Lambda = \{p_0, p_1, p_2, \cdots, p_n\}$ is an optimal full path, then the sub-tree which is rooted at $p_i(0 \leq i \leq n)$ is an optimal i-subtree.

Based on the *S-C* tree model, the IDS problem can be converted as: given a dataflow F and its *S-C* tree $Tree^F$, find an optimal full path of $Tree^F$.

3.2 Proofs of the Theorems

For convenience, given an *S-C* tree T, assume that a node $\tau \in N|_T^k$, we use $subtree(\tau)$ to denote a k-subtree which is rooted at τ.

Definition 9. *Given an* S-C *tree, let τ and τ' be the nodes at the same level. We say that τ is* **equivalent to** *τ', denoted as $\tau \equiv \tau'$, if and only if $((\tau, \tau' \in S) \vee (\tau, \tau' \in C)) \wedge (cost(\tau) = cost(\tau'))$. If $((\tau, \tau' \in S) \vee (\tau, \tau' \in C)) \wedge (cost(\tau) < cost(\tau'))$, we say that τ is* **superior to** *τ', denoted as $\tau \prec \tau'$.*

The equivalent and superior relations between nodes are both transitive.

Lemma 1. *Given a dataflow $F = \{d_0, d_1, \ldots, d_n\}$ and its* S-C *tree $Tree^F$, if τ and τ' are both S-nodes at the same level, then $\tau \equiv \tau'$.*

Proof. Assume that $\tau, \tau' \in N|_{Tree^F}^i, 0 < i \leq n$. Since τ and τ' are both S-nodes, according to Definition 5, $cost(\tau) = cost(\tau') = x(d_i)$, thus $\tau \equiv \tau'$. $\square$

Definition 10. *Assume that δ and δ' are k-subtrees, τ and τ' are nodes belonging to δ and δ' respectively. We say that τ' is the* **corresponding node** *of τ about δ and δ', denoted as $\tau \leftrightarrow \tau'|(\delta, \delta')$, if one of the following conditions is satisfied:*

(1) $(\delta = subtree(\tau)) \wedge (\delta' = subtree(\tau'))$;
(2) $\exists(\upsilon \leftrightarrow \upsilon'|(\delta, \delta')) \wedge (\tau = left(\upsilon)) \wedge (\tau' = left(\upsilon'))$;
(3) $\exists(\upsilon \leftrightarrow \upsilon'|(\delta, \delta')) \wedge (\tau = right(\upsilon)) \wedge (\tau' = right(\upsilon'))$.

Definition 11. *Assume that δ and δ' are both k-subtrees, Λ and Λ' are full path of δ and δ' respectively. Let $\tau_i \in (N|_\delta^i \cap \Lambda)$ and $\tau_i' \in (N|_{\delta'}^i \cap \Lambda')$. We say that Λ' is the* **corresponding path** *of Λ about δ and δ', denoted as $\Lambda \leftrightarrow \Lambda'|(\delta, \delta')$, if and only if $\tau_i \leftrightarrow \tau_i'|(\delta, \delta')$ for any $0 \leq i \leq k$.*

In Fig. 3, $\{s_1, c_2, s_5, c_{10}\} \leftrightarrow \{c_1, c_3, s_7, c_{14}\}|(subtree(s_1), (subtree(c_1))$.

Definition 12. *Assume that δ and δ' are both k-subtrees, we say that δ is* **equivalent to** *δ', denoted as $\delta \equiv \delta'$, if and only if for each node τ in δ and its corresponding node τ' in $\delta', \tau \equiv \tau'$. We say that δ is* **superior to** *δ', denoted as $\delta \prec \delta'$, if and only if the set of nodes of δ can be separated into two subsets, A and B, which satisfy the following conditions:*

(1) $A = \{\tau|(\tau \leftrightarrow \tau'|(\delta, \delta')) \wedge (\tau \equiv \tau')\}$;
(2) $B = \{\tau|(\tau \leftrightarrow \tau'|(\delta, \delta')) \wedge (\tau \prec \tau')\}$;
(3) B is nonempty.

That is, each node in A is equivalent to its corresponding node in δ', and each node in B is superior to its corresponding node in δ'. The equivalent and superior relations between k-subtrees are both transitive.

Definition 13. *Given an* S-C *tree T, let $P_1^k = \{p_{1,0}, p_{1,1}, \ldots, p_{1,k}\}$ and $P_2^k = \{p_{2,0}, p_{2,1}, \ldots, p_{2,k}\}$ be k-paths of $T, 0 < k \leq n$. We say that P_1^k is* **equivalent to** *P_2^k, denoted as $P_1^k \equiv P_2^k$, if and only if $p_{1,i} \equiv p_{2,i}$ for any $0 \leq i \leq n$. We say that P_1^k is* **superior to** *P_2^k, denoted as $P_1^k \prec P_2^k$, if and only if P_1^k can be separated into two nonempty subsets, $P_1^k = A \cup B$, such that $A = \{p_{1,i}|p_{1,i} \equiv p_{2,i}, 0 \leq i < n\}$ and $B = \{p_{1,i}|p_{1,i} \prec p_{2,i}, 0 \leq i < n\}$.*

The equivalent and superior relations between k-paths are also transitive.

Lemma 2. *Given a workflow $F = \{d_0, d_1, \ldots, d_n\}$ and its* S-C *tree $Tree^F$, let $\tau, \tau' \in N|_{Tree^F}^i, 0 < i \leq n$, if $\tau \equiv \tau'$, then $subtree(\tau) \equiv subtree(\tau')$.*

Proof. Since $\tau, \tau' \in N|_{Tree^F}^i$, and $left(\tau)$ and $left(\tau')$ are both S-nodes, based on Lemma 1, we have: $left(\tau) \equiv left(\tau')$. (a-1)

As right(τ) and right(τ') are both C-nodes, $cost(right(\tau)) = \sum_{\alpha \in Path^{\mapsto}_{right(\tau)}} \omega(\alpha) = \sum_{\alpha \in Path^{\mapsto}_{\tau}} \omega(\alpha) + \omega(right(\tau)) = cost(\tau) + y(d_{i+1})$, and $cost(right(\tau')) = \sum_{\alpha \in Path^{\mapsto}_{right(\tau')}} \omega(\alpha) = \sum_{\alpha \in Path^{\mapsto}_{\tau'}} \omega(\alpha) + \omega(right(\tau')) = cost(\tau') + y(d_{i+1})$. Due to $\tau \equiv \tau'$, we have $cost(\tau) = cost(\tau')$, so $cost(right(\tau)) = cost(right(\tau'))$, then: $right(\tau) \equiv right(\tau')$. (a-2)

Summarizing (a-1) and (a-2), both $left(\tau)$ and $right(\tau)$ are respectively equivalent to $left(\tau')$ and $right(\tau')$. By this analogy, the rest nodes of $subtree(\tau)$ and $subtree(\tau')$ can be dealt with in the same manner. Hence, any node of $subtree(\tau)$ is equivalent to its corresponding node of $subtree(\tau')$. That is: $subtree(\tau) \equiv subtree(\tau')$. □

Corollary 1. *Let τ and τ' be two nodes at the same level in an* S-C *tree T, then $subtree(left(\tau)) \equiv subtree(left(\tau'))$.*

Proof. Assume that $\tau, \tau' \in N|_T^i$, then $left(\tau) \in N|_T^{i+1}$ and $left(\tau') \in N|_T^{i+1}$. According to Lemma 1, $left(\tau) \equiv left(\tau')$. Based on Lemma 2, we can obtain: $subtree(left(\tau)) \equiv subtree(left(\tau'))$. □

Lemma 3. *Let τ and τ' be two nodes at the same level in an* S-C *tree T, if $\tau, \tau' \in C$ and $\tau \prec \tau'$, then $subtree(\tau) \prec subtree(\tau')$.*

Proof. Assume that $\tau, \tau' \in N|_T^i$. Based on Lemma 1, we have:

$left(\tau) \equiv left(\tau')$. (b-1)

Similar to the proof of Lemma 2, we have: $cost(right(\tau)) = \sum_{\alpha \in Path^{\mapsto}_{right(\tau)}} \omega(\alpha) = \sum_{\alpha \in Path^{\mapsto}_{\tau}} \omega(\alpha) + \omega(right(\tau)) = cost(\tau) + y(d_{i+1})$, and $cost(right(\tau')) = \sum_{\alpha \in Path^{\mapsto}_{right(\tau')}} \omega(\alpha) = \sum_{\alpha \in Path^{\mapsto}_{\tau'}} \omega(\alpha) + \omega(right(\tau')) = cost(\tau') + y(d_{i+1})$. Due to $\tau \prec \tau'$, we have $cost(\tau) < cost(\tau')$, thus $cost(right(\tau)) < cost(right(\tau'))$, then: $right(\tau) \prec right(\tau')$. (b-2)

Summarizing (b-1) and (b-2), we can separate the set of nodes of $subtree(\tau)$ into two subsets, A and B. We add the nodes of $subtree(left(\tau))$ into subset A, and $right(\tau)$ into subset B. By this analogy, $right(\tau)$ can be dealt with in the same manner like τ until all the nodes are contained in A or B. Each node in A is equivalent to its corresponding node of $subtree(\tau')$, and each node in B is superior to its corresponding node of $subtree(\tau')$. Therefore, $subtree(\tau) \prec subtree(\tau')$. □

Theorem 1. *Given an* S-C *tree T, let $P_i^k = \{p_{i,0}, p_{i,1}, p_{i,2}, \dots, p_{i,k}\}$ and $P_j^k = \{p_{j,0}, p_{j,1}, p_{j,2}, \dots, p_{j,k}\}$ be any two k-paths of $T, i \neq j$. If $Cost(P_j^k) < Cost(P_i^k)$, then $subtree(left(p_{i,k}))$ is not an optimal $(k+1)$-subtree.*

Proof. Following Corollary 1, we have $subtree(left(p_{i,k})) \equiv subtree(left(p_{j,k}))$. Assume that P is the optimal full path of $subtree(left(p_{i,k}))$, as shown in Fig. 4, there must exist a corresponding path P' in $subtree(left(p_{j,k}))$ which satisfies $P' \equiv P$. Since $Cost(P_j^k) < Cost(P_i^k)$, then the full path $link(P_j^k, P') \prec link(P_i^k, P)$. Thus $link(P_i^k, P)$ must not be the optimal full path of T. That is, $subtree(left(p_{i,k}))$ is not the sub-tree through which the optimal full path passes. Hence, $subtree(left(\tau))$ is not an optimal $(k+1)$-subtree. □

Theorem 2. *Given an* S-C *tree of a dataflow $F = \{d_0, d_1, \dots, d_n\}$, let $\Omega = \{P_1^k, P_2^k, \dots, P_m^k\}$ be a set of k-paths, where $P_i^k = \{p_{i,0}, p_{i,1}, p_{i,2}, \dots, p_{i,k}\}, 1 \leq$*

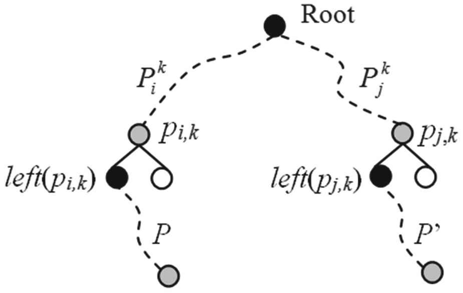


Fig. 4. An exemplar graph of Theorem 1.

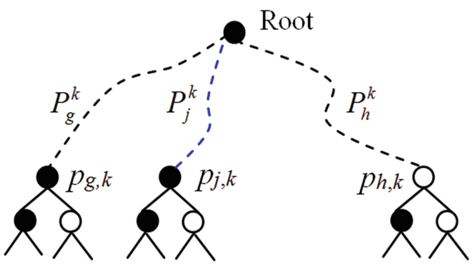


Fig. 5. An exemplar graph of Theorem 2.

$i \leq m$. Assume that Ω contains an optimal full path from the root to the k^{th} level, then if $Cost(P_j^k) = \min_{p^k \in \Omega} Cost(P^k)$ and $p_{j,k} \in S$, $subtree(p_{j,k})$ must be an optimal k-subtree.

Proof. As shown in Fig. 5, let P_g^k and P_h^k $(g \neq h \neq i)$ be any two k-paths taken from Ω such that $p_{g,k} \in S$ and $p_{h,k} \in C$. According to the precondition, we have $Cost(P_j^k) < Cost(P_g^k)$ and $Cost(P_j^k) < Cost(P_h^k)$.

(1) Since $p_{g,k}$, $p_{j,k} \in S$, based on Lemmas 1 and 2, we have $p_{g,k} \equiv p_{j,k}$, thus $subtree(p_{j,k}) \equiv subtree(p_{g,k})$. Let P be the optimal full path of $subtree(p_{g,k})$, then there must exist a corresponding path P' in $subtree(p_{j,k})$, which satisfies $P' \equiv P$. Since $Cost(P_j^k) < Cost(P_g^k)$, we have $link(P_j^k, P') \prec link(P_g^k, P)$. Thus $link(P_g^k, P)$ must not be the optimal full path of $Tree^F$. That is, $subtree(p_{g,k})$ is not the sub-tree which the optimal full path passes through, thus $subtree(p_{g,k})$ is not an optimal k-subtree.

(2) Based on Corollary 1, we have:

$$subtree(left(p_{j,k})) \equiv subtree(left(p_{h,k})). \tag{c-1}$$

Since $P_{j,k} \in S$, thus $cost(right(p_{j,k})) = y(d_{i+1})$. While due to $P_{h,k} \in C$, we also have $cost(right(p_{h,k})) = \sum_{\alpha \in Path^{\mapsto}_{right(P_{h,k})}} \omega(\alpha)$ which is equal to $\sum_{\alpha \in Path^{\mapsto}_{P_{h,k}}} \omega(\alpha) + \omega(right(p_{h,k})) = cost(p_{h,k}) + y(d_{k+1})$. That is, $subtree(right(p_{j,k})) \prec subtree(right(p_{h,k}))$. As $right(p_{j,k})$ and $right(p_{h,k})$ are both C-nodes, according to Lemma 3, we have:

$$subtree(right(p_{j,k})) \prec subtree(right(p_{h,k})). \tag{c-2}$$

Due to (c-1), (c-2) and $Cost(P_j^k) < Cost(P_h^k)$, we can obtain that, assuming P is the optimal full path of subtree($p_{h,k}$), there must exist a corresponding path P' in $subtree(p_{j,k})$, which satisfies $P' \prec P$, so we have $link(P_j^k, P') \prec link(P_h^k, P)$. Hence, $subtree(p_{h,k})$ is not the sub-tree through which the optimal full path of $tree^F$ passes, that is, $subtree(p_{g,k})$ is not an optimal k-subtree.

Summarizing (1) and (2), if Ω contains an optimal full path from the root to the k^{th} level, the optimal full path must pass through $subtree(p_{j,k})$, thus $subtree(p_{j,k})$ must be an optimal k-subtree.

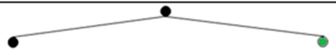
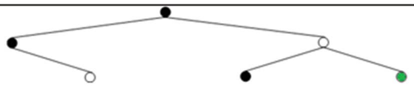
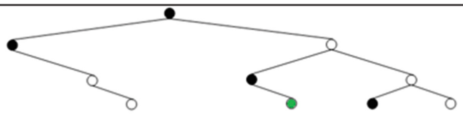
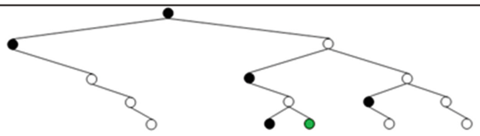
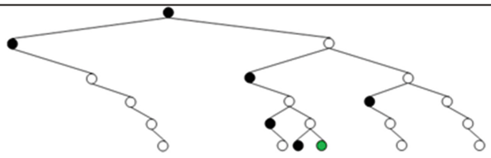
4 Algorithm for the IDS Problem Based on the S-C Tree

Based on the *S-C* tree model, the IDS problem is converted to searching an optimal full path of a given dataflow *S-C* tree. Using Theorems 1 and 2, we can obtain the following pruning strategies. By these strategies, the search space can be greatly reduced.

(1) Search for the optimal full path of the given *S-C* tree from top to bottom by level. At each level k, the search space is set to $\Omega = \{P_1^k, P_2^k, \ldots, P_m^k\}$, in which $P_i^k = \{p_{i,0}, p_{i,1}, p_{i,2}, \ldots, p_{i,k}\}, 1 \le i \le m$, is the k-path that has not been pruned off.

(2) At each level k, let P_j^k be the current best k-path which satisfies $Cost(P_j^k) = \min_{p^k \in \Omega} Cost(P^k)$, then:

   (a) If $p_{j,k} \in C$, based on Theorem 1, for any $p_{i,k}, i \ne j, subtree(left(p_{i,k}))$ is not an optimal $(k+1)$-subtree thus can be pruned off, so all

Level k	$\|\Omega\|_k$	Exemplar graph of search process (The node in green expresses $p_{j,k}$, which is the last node of the current best k-path P_j^k in Ω.)
0	1	
1	2	
2	3	
3	4	
4	5	
5	6	

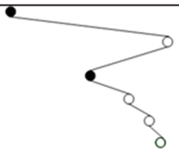
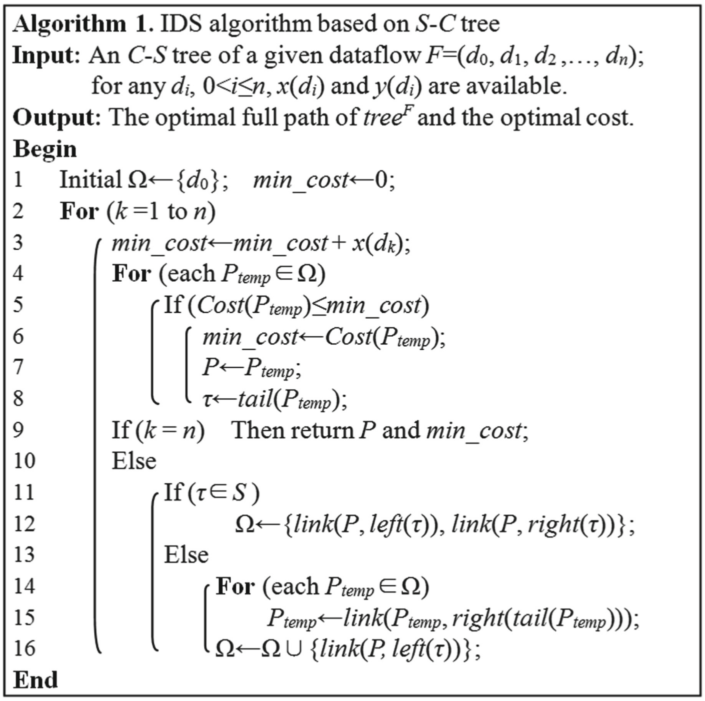
The optimal full path: 

Fig. 6. An example of Algorithm 1.



```
Algorithm 1. IDS algorithm based on S-C tree
Input: An C-S tree of a given dataflow $F=(d_0, d_1, d_2, \ldots, d_n)$;
       for any $d_i$, $0<i\leq n$, $x(d_i)$ and $y(d_i)$ are available.
Output: The optimal full path of $tree^F$ and the optimal cost.
Begin
1  Initial $\Omega \leftarrow \{d_0\}$;   $min\_cost \leftarrow 0$;
2  For ($k$ =1 to $n$)
3      $min\_cost \leftarrow min\_cost + x(d_k)$;
4      For (each $P_{temp} \in \Omega$)
5          If ($Cost(P_{temp}) \leq min\_cost$)
6              $min\_cost \leftarrow Cost(P_{temp})$;
7              $P \leftarrow P_{temp}$;
8              $\tau \leftarrow tail(P_{temp})$;
9      If ($k = n$)   Then return $P$ and $min\_cost$;
10     Else
11         If ($\tau \in S$)
12                 $\Omega \leftarrow \{link(P, left(\tau)), link(P, right(\tau))\}$;
13         Else
14             For (each $P_{temp} \in \Omega$)
15                 $P_{temp} \leftarrow link(P_{temp}, right(tail(P_{temp})))$;
16             $\Omega \leftarrow \Omega \cup \{link(P, left(\tau))\}$;
End
```

Fig. 7. The IDS algorithm based on S-C tree.

$link(P_i^k, right(p_{i,k}))$, $i \neq j$, as well as $link(P_j^k, left(p_{j,k}))$ and $link(P_j^k, right(p_{j,k}))$ will be contained in Ω for the next round of search.

(b) If $p_{j,k} \in S$, according to Theorem 2, $subtree(p_{j,k})$ is the optimal k-subtree, so any $subtree(p_{i,k}), i \neq j$, can be pruned off, so only $link(P_j^k, left(p_{j,k}))$ and $link(P_j^k, right(p_{j,k}))$ can be contained into Ω for the next round of search.

Figure 6 shows an example of the searching process. We can see that more than $\frac{k-1}{2k}$ of the branches are pruned off at each level of searching. Based on the strategies above, we present the IDS algorithm in Fig. 7.

In line 8 and 15, the function $tail(P_{temp})$ means the last node of P_{temp}.

Theorem 3. *The searching space Ω increases linearly with the level of the* S-C *tree in Algorithm 1.*

Proof. Let $|\Omega|_k$ denote the size of Ω in the searching of the k^{th} level, $0 \leq k \leq n$. According to Algorithm 1, we have:

(1) When $k = 0, \Omega = \{d_0\}$, thus $|\Omega|_0 = 1$.
(2) When $k > 0$, if $\tau \in S, |\Omega|_{k+1} = 2$; else $\tau \in C$, then for each $P_{temp} \in \Omega, P_{temp}$ will be replaced by $link(P_{temp}, right(tail(P_{temp})))$, and $link(P, left(\tau))$ will

be the only additional new comer of Ω in the next round of searching, hence, $|\Omega|_{k+1} = |\Omega|_k + 1$.

Therefore, in the worst-case, $|\Omega|_{k+1} = |\Omega|_k + 1$, then we have $|\Omega|_k = |\Omega|_{k-1} + 1$, $|\Omega|_{k-1} = |\Omega|_{k-2} + 1, \ldots$, $|\Omega|_1 = |\Omega|_0 + 1$, so we can obtain that $|\Omega|_k = k + 1$. That is, Ω increases linearly with the level of the *S-C* tree. □

For a k-path $P^k = \{p_0, p_1, p_2, \ldots, p_k\}, 0 \leq k \leq n$, the calculation of $Cost(P^k)$ takes $O(k)$ time. Following Theorem 3, Algorithm 1 takes $O(n^3)$ time in the worst case. Furthermore, since Ω is composed of n n-paths, thus the space complexity of Algorithm 1 is $O(n^2)$.

5 Conclusions

In this paper, we solved the IDS problem for linear-structure dataflow by using an *S-C* tree model. The running time of our algorithm is $O(n^3)$, which improves the previous bound of $O(n^4)$. In the near future, we will study the IDS problems for cloud dataflow with a non-linear structure, such as parallel structure and non-structure dataflows.

Acknowledgements. This paper is supported by national natural science foundation of China: 61472222, and natural science foundation of Shandong province: ZR2012Z002.

References

1. Deelman, E., Chervenak, A.: Data management challenges of data-intensive scientific workflows. In: IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), pp. 687–692, Lyon, France (2008)
2. Yuan, D., Yang, Y., Liu, X., Zhang, G., Chen, J.: On-demand minimum cost benchmarking for intermediate data storage in scientific cloud workflow systems. J. Parallel Distrib. Comput. **71**(2), 316–332 (2011)
3. Adams, I., Long, D.D.E., Miller, E.L., Pasupathy, S., Storer, M.W.: Maximizing efficiency by trading storage for computation. In: Workshop on Hot Topics in Cloud Computing (HotCloud 2009), pp. 1–5, San Diego, CA (2009)
4. Yuan, D., Yang, Y., Liu, X., Zhang, G., Chen, J.: A data dependency based strategy for intermediate data storage in scientific cloud workflow systems. Concurr. Comput. Pract. Exp. **24**(9), 956–976 (2010)
5. Zohrevandi, M., Bazzi, R.A.: The bounded data reuse problem in scientific workflows. In: 2013 IEEE 27th International Symposium on Parallel & Distributed Processing, pp. 1051–1062 (2013)
6. Han, L.X., Xie, Z., Baldock, R.: Automatic data reuse for accelerating data intensive applications in the Cloud. In: The 8th International Conference for Internet Technology and Secured Transactions (ICITST-2013), pp. 596–600 (2013)