Theoretical Computer Science 657 (2017) 48–53

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Improved algorithms for intermediate dataset storage in a cloud-based dataflow

Jie Cheng [a], Daming Zhu [b,*], Binhai Zhu [c]

[a] School of Mechanical, Electrical and Information Engineering, Shandong University, Weihai, China
[b] School of Computer Science and Technology, Shandong University, Jinan, China
[c] Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA

ARTICLE INFO

Article history:
Received 17 December 2015
Received in revised form 11 May 2016
Accepted 19 May 2016
Available online 6 June 2016

Keywords:
Algorithm
Complexity
Dataflow
Dataset
Cloud computing

ABSTRACT

In order to run a dataflow with as low cost as possible, it is often faced with deciding which data-sets in a data-set sequence should be stored, with the rest regenerated. The Intermediate Data-set Storage problem arises from this situation. The current best algorithm for this problem takes $O(n^4)$ time. In this paper, we present two improved algorithms for this problem, the first of which can achieve a time complexity $O(n^2)$, the second of which $O(rn)$, where n is the number of data-sets in a dataflow, r is a numerical number which indicates how large it is for the maximum storage cost to be divided by the minimum computation cost in the dataflow.

1. Introduction

A cloud-based dataflow is a data driven workflow deployed in a cloud computing environment. A dataflow has an initial data-set, an output data-set, and a group of intermediate data-sets generated during the execution. The intermediate data-sets often contain valuable intermediate results, thus would be frequently traced back for reanalyzing or reusing [1]. Since the dataflow systems are executed in a cloud computing environment, all used resources need to be paid for. As indicated in [2], storing all of the intermediate data-sets may induce a high storage cost, while if all the intermediate data-sets are deleted and regenerated when needed, the computation cost of the system may also be very high. Hence, a strategy is needed for selecting to store some data-sets and regenerate the rest of them when needed, so as to lower the total cost of the whole workflow system [3,4]. This leads to the intermediate data-set storage problem (abbr. IDS).

In a cloud dataflow system, when a deleted data-set needs to be regenerated, the computation cost will involve not only itself but its direct predecessor. If its predecessor is also deleted, the computation cost of a sequence of deleted data-sets needs to be accumulated. In [2], Yuan et al. presented the background of IDS in scientific workflows and proposed an intermediate data dependency graph. Based on this kind of graphs, they proposed two algorithms as the minimum cost benchmark of IDS, an algorithm for linear IDS which takes $O(n^4)$ time, as well as an algorithm for parallel IDS which takes $O(n^9)$ time. A preliminary version of the paper presented in FAW2015 has improved the algorithm for linear IDS to run in $O(n^3)$ time [5].

Actually, there have been many related approaches for deciding which intermediate datasets should be stored. Zohrevandi and Bazzi [6] presented a branch-and-bound algorithm for the common intermediate data-set storage between two scientific

* Corresponding author.
E-mail addresses: chjie@sdu.edu.cn (J. Cheng), dmzhu@sdu.edu.cn (D. Zhu), bhz@cs.montana.edu (B. Zhu).

http://dx.doi.org/10.1016/j.tcs.2016.05.042
0304-3975/

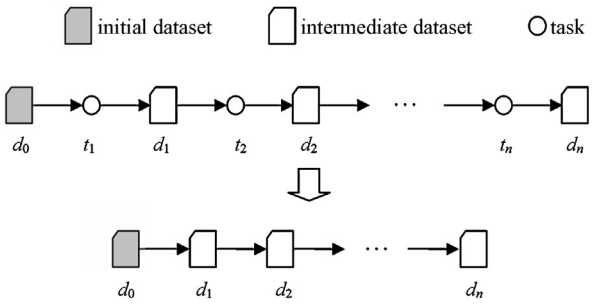

Fig. 1. The exemplar graph of a linear dataflow, which has been presented in [5] originally.

workflows. Adams et al. [3] proposed a model which can balance the computation cost and the storage cost. Han et al. proposed a Perti-nets based method [7] to support automatic intermediate data reusing for large-scale cloud dataflows.

In this paper, we devote to improve the time complexity for solving linear IDS. We set a rooted binary tree to represent the feasible solutions of an IDS instance. Using this binary tree, a dynamic programming algorithm with $O(n^2)$ time can be designed, where n is the data-set number in a dataflow. A more effective dynamic programming algorithm with $O(rn)$ time can also be designed, where r is the numerical number of the maximum storage cost over all data-sets in a dataflow divided by the minimum computation cost over all data-sets in that dataflow.

2. The intermediate dataset storage problem

Let $T = [t_1, ..., t_n]$ be a linear structured system of n processing units. Then a dataflow stems from the consecutive data-set processing on T, and has usually been represented by a data-set sequence $F = [d_0, d_1, ..., d_n]$, where d_i is produced from t_i which uses d_{i-1} as its input data, $1 \le i \le n$. A dataflow with n intermediate processing units is depicted in Fig. 1.

A data-set can be processed in an action of either *storage* or *computation* [1], where the action for d_0 in F must be storage. Processing a data-set always takes resources. The resources for processing a data-set in storage can be measured by its *size*, while the resources for processing a data-set in computation can be measured by its *instance-hours*. Uniformly, we use the so called *cost* to represent how many resources it can take to process a data-set. In the situation a data-set d_j is demanded, d_j is available if it is in storage; or should be regenerated if it is in computation. To regenerate d_j which is in computation, one has to trace back to the closest data-set in storage previous to d_j.

A *processing sequence* of F refers to a sequence of storage or computation actions for processing the data-sets in the order as they are in F, where by default, the action for d_0 is storage. Let $p = [p(d_0), p(d_1), ..., p(d_n)]$ be a processing sequence of F with $p(d_i) \in \{S, C\}$, where S, C stand for storage and computation respectively. We refer to d_i as the *s-prior* of d_j in p, $0 \le i < j$, if $p(d_i) = S$ and for k with $i < k < j$, $p(d_k) = C$. Let $x(d_j)$, $y(d_j)$ for $0 \le j \le n$ be the cost for processing d_j in storage and computation respectively. Then with respect to p, the *generating cost* of d_j is formulated by,

$$cost(d_j) = \begin{cases} x(d_j), & p(d_j) = S; \\ \sum_{k=i+1}^{j} y(d_k), & p(d_j) = C, d_i \text{ is the s-prior of } d_j \text{ in } p. \end{cases} \tag{2.1}$$

The generating cost summation over all data-sets in F with respect to p is referred to as the *cost* of p. Since a data-set is processed in either storage or computation, it is of great concern for which processing sequence of F has as low cost as possible. Thus the Intermediate Data-set Storage problem, IDS namely, is given by a dataflow whose data-sets each has a cost in storage and a cost in computation, asks to find a processing sequence, such that the cost of it is minimized. The problem can be formalized as,

Instance: A dataflow $F = [d_0, d_1, \cdots, d_n]$, the cost of processing d_i in storage and computation are $x(d_i)$ and $y(d_i)$ respectively.

Question: Find a processing sequence of F, whose cost is minimized over all processing sequences of F.

A processing sequence of F is *optimal*, if its cost is minimized.

3. Dynamic programming on a binary tree

Let $F[i] = [d_0, ..., d_i]$, $1 \le i \le n$. We will employ a dynamic programming for i going from 1 to n, to enumerate all those processing sequences of $F[i]$ which can grow up into an optimal processing sequence of F. Toward this purpose, we propose a binary tree to maintain those promising processing sequences of $F[i]$.