Cluster Comput (2015) 18:1157–1169
DOI 10.1007/s10586-015-0467-3

A data transmission algorithm for distributed computing system based on maximum flow

Xiaolu Zhang[1] · **Jiafu Jiang**[1] · **Xiaotong Zhang**[1] · **Xuan Wang**[1]

Received: 29 March 2014 / Revised: 20 May 2015 / Accepted: 1 June 2015 / Published online: 15 July 2015

Abstract Data skew can lead to load imbalance and longer computation time in the distributed computing system. To avoid data skew and reduce the data computation time, it is necessary to transmit the data to appropriate machines, this may however take too much network resources. How to balance the computational resources and the network resources is a problem. In this paper, we introduce a computation model called distributed two-phase model, in which the process of a task can be divided into two independent phases: data transmission and data computation. Suppose an upper bound of relative computation time is given, we show how to schedule data transmission with minimum resources, such as data transmission time and occupied bandwidth, to meet the demand. In this paper, we present a novel algorithm to minimize data transmission time and network bandwidth usage in the data transmission phase, with the conditions that an upper bound of relative computation time of data computation phase is given. Moreover, the number of nodes that participate in data computation phase is also reduced, in this way, the computational resources are saved. The simulation results show that the occupied bandwidth can be reduced effectively (about 70 %) in the situation of large-scale data sets and large number of nodes. Our algorithm is also shown to be available in replication situation.

✉ Xiaotong Zhang
zxt@ies.ustb.edu.cn

Xiaolu Zhang
zxl0714@163.com

Jiafu Jiang
jiangjiafu1989@gmail.com

Xuan Wang
ustb_wx@163.com

[1] University of Science and Technology Beijing, Beijing, China

Keywords Distributed computing system · Minimize bandwidth usage · Data transmission time

1 Introduction

Data skew can result in load imbalance and longer computation time in the distributed computing system. To avoid data skew, it is necessary to transmit the data to appropriate machines and process the data concurrently. Data skew has been widely studied in the literature [25], and a large number of skew aware load balancing algorithms have been developed [15,22,24], some of which are quite successful. However, none of these algorithms considers the bandwidth usage. Resources of a distributed system can be divided into computational resources and network resources. Due to the rapid development of the microprocessor, the effects of the computation delay on the overall performance of the system are becoming less significant [32]. On the other hand, the communication delay still has a major effect on the system performance and can be a major problem especially in a system with many nodes and a high communication frequency [33]. Making the data distributed evenly across a cluster will take too much network resources. Therefore the trade-off between the computational resources and the network resources has to be considered.

Suppose an upper bound of relative computation time is given, there can be a reasonable data transmission path so that the computation time will not exceed the given upper bound. Although different distributed systems may have different frameworks, most of them have the same model, which contains two basic processes: data transmission and data computation. This model has been used in many recent distributed systems such as Dryad [19] and Google MapReduce [8]. To simplify the model, two rules are added to it: (1) data

computation phase of a task cannot start until all the data of the task are transmitted to the appropriate machines; (2) the data in different machines can be processed concurrently without any influence on each other. The first rule seems to be too strong, but in fact, there are always many tasks in a distributed system, therefore the computational resources will not be wasted: when task A is in data transmission phase, task B may be in data computation phase. By separating the data transmission phase and the data computation phase, we can take full advantage of the computational resources and the network resources of the distributed system.

Data skew has been widely studied in the literature, and a large number of skew aware load balancing algorithms have been developed [15,24], some of which are quite successful. In recent years, with the development of MapReduce, some algorithms that focus on solving the data skew in MapReduce are presented, such as [14,17,29]. Hadoop uses JobQueueTaskScheduler to lower the cost of network traffic of data transmission, the JobQueueTaskScheduler implements a greedy method that transfers the data block to the nearest node. However, none of these algorithms considers the trade-off between the computational resources and the network resources. From the above facts, we introduce our algorithm, which dynamically selects the data transmission path which can be best suited for a particular task, and reduces the bandwidth usage based on the dynamically measured resource parameters. Our algorithm is based on the distributed two-phase model (hereafter referred to as the DTPM). DTPM is different from MapReduce. Both the map phase and the reduce phase of MapReduce contain data transmission and data computation, while in DPTM, data transmission and computation are separated. In some ways, DTPM is similar to the Map phase of MapReduce model. The basic unit of data storage, data transmission, and data computation in DTPM is the block. Each block has the same size. We use relative time instead of absolute time in our algorithm, for example, if β stands for the time node A uses to process one block of data, and node B is two times faster that node A, then node B takes 0.5β to process the block. This paper is concerned with reducing of communication bandwidth usage and data transmission time in the DTPM. The two major contributions of this study are:

- We propose an algorithm to accomplish tasks in DTPM with minimum data transmission time and communication bandwidth usage, in the condition that the relative computation time does not exceed a given upper bound.
- We reduce the number of nodes which participate in the data computation phase.

This paper is organized as follows. The model description is given in Sect. 2, in which we will introduce the background, the data transmission model and the methods to establish the flow network and find out the minimum data transmission time. We discuss how to minimize the occupied bandwidth and number of nodes in Sect. 3. We show how our algorithm can be available in replication situation in Sect. 4. The emulation results are shown in Sect. 5. In Sect. 6, we give a general introduction and comparison of related works. In Sect. 7 we summarize our results, and mention some future work.

2 Model description

2.1 Background

A computer cluster consists of numerous network devices (routers and switches) and computers. The computers in the cluster are heterogeneous, which means that different nodes have different compute ability, storage capacity, and bandwidth. A simple network of computer cluster is shown in Fig. 1 (network devices are omitted). There are N nodes in the Fig. 1: node 1 $\sim$ node N. The numbers placed on the arcs represent the network bandwidth between two compute nodes. In order to describe DTPM and discuss the problem in this paper, more terminology are needed. Table 1 summarizes some of the important terminology that will be used in this paper.

In the distributed system, data are distributed on the computer cluster. When a task begins, some data which are related to the task will be transmitted from one node to another and then each node will process the data that are stored on it. The symbol M_i stands for the initial amount of data of node i before data transmission phase, and D_i stands for the final amount of data of node i after the data transmission phase. Note that D_i is not a given parameter, the value of D_i is unknown until the data transmission phase is finished. The symbol F_i is a Boolean value. If F_i is true, then node i is busy and cannot process any data except that it can transmit the data of it to other idle nodes. Otherwise, node i is idle and it can process more data including data transmission and data computation. Most of the properties that used in our algorithm are changing all the time. Therefore, before each task is executed, the properties will be updated to latest, in this way multitask can be supported . Similarly, MapReduce

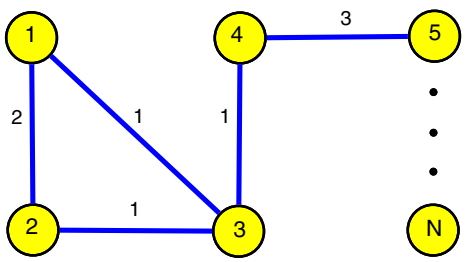

Fig. 1 A simple network of computer cluster

Table 1 Some terminology

Parameter	Description
P_i	Processing speed of node i
R_i	Maximal remaining storage capacity of node i
C_i	Maximal transmission rate of node i
M_i	Amount of initial data on node i
D_i	Amount of final data on node i
Q_i	Amount of data that node i should transmit
F_i	A busy flag of node i
N	Total number of nodes
T_{tran}	Relative total transmission time
T_{comp}	Relative total computation time
T_{max}	The upper bound of T_{comp}
T_{task}	Total execution time of a task
T_i	The relative time that node i takes to finish processing its data
α	Basic time unit in the data transmission phase
β	Basic time unit in the data computation phase

achieves multiple tasks by maintaining the information (slots, health status, etc.) of all of the slave nodes. There are many methods [3,18,26] that can be used to monitor the status of the cluster, however it is not our theme in this paper. As the executing process of a task can be divided into two phases:

Step 1 *Data Transmission* Different nodes have different abilities (processing speed, remaining storage capacity, transmission rate etc.), therefore nodes that have no enough ability to process data should transmit their data to other nodes.

Step 2 *Data Computation* In this step, some operations that specified by the task are performed on the data concurrently.

Then the total processing time of a task is:

$$T_{task} = \alpha T_{tran} + \beta T_{comp} \tag{1}$$

Since T_{tran} and T_{comp} are both relative value, the factors α and β stand for the basic time unit of the data transmission phase and the data computation phase, respectively. We need α and β because that the time unit in data transmission phase and data computation phase may be different. For example, if $T_{tran} = 3$, $T_{comp} = 5$, $\alpha = 2second$, $\beta = 1second$, then $T_{task} = 2 \times 3 + 1 \times 5 = 11(second)$. In fact α and β are just constants that are used to unify the time unit of the two phases, the accurate values of them are not important, because they have no effect on our algorithm. Both α and β vary in different tasks, and they are difficult to estimate because the absolute speed of data computation and data transmission of each node are difficult to estimate. Let T_i be the relative time that node i takes to finish processing the data on it:

$$T_i = D_i / P_i \tag{2}$$

We suppose that T_i grows linearly as the amount of data increases for simplicity. However, more complicated relation of T_i, D_i, and P_i is supported. In fact, we just need to know how T_i is related to D_i and P_i. For example, let $T_i = f(D_i, P_i)$, if f is known, our algorithm can work fine. The algorithm description and the examples in this paper are based on the simplified model. The relative total computation time T_{comp} depends on when the last node finishes the computation phase. Therefore,

$$T_{comp} = max(T_i) \tag{3}$$

Instead of finding the minimum T_{task}, we focus on finding the minimum T_{tran} and communication bandwidth usage while satisfying the T_{comp} restriction ($T_{comp} \leq T_{max}$). In fact, it is difficult to figure out the minimum T_{task}. To figure out the minimum T_{task}, the absolute speed of data computation and data transmission of each node has to be known. Besides, both α and β vary in different tasks, and they are difficult to estimate.

2.2 Data transmission

The nodes in the cluster vary in computation speed, storage capacity, bandwidth, etc., therefore, nodes that have more capacity should undertake more work. Conversely, nodes that have lower capacity should deliver their data to other nodes which are more powerful. Let Q_i stand for the amount of data that node i should transmit. If Q_i is positive, it indicates the amount of data that node i should deliver to others. If Q_i is negative, $|Q_i|$ indicates the amount of data that node i can receive besides of the initial data on it.

$$Q_i = \begin{cases} M_i & F_i = True \\ max(\lceil M_i - T_{max} P_i \rceil, -\lfloor R_i \rfloor) & F_i = False \end{cases} \quad (4)$$

If node i is busy ($F_i = True$), it should transmit all its data to other nodes, in this case, the amount of data that node i should transmit is M_i. Otherwise, if node i is not busy ($F_i = False$), $Q_i = max(\lceil M_i - T_{max} P_i \rceil, -\lfloor R_i \rfloor)$. If $M_i - T_{max} P_i$ is greater than 0, then $Q_i = \lceil M_i - T_{max} P_i \rceil$, which means that node i can only process part of its initial data, and it has to transmit the remain of its data to other nodes. If $M_i - T_{max} P_i$ is less that 0, it means that node i can process more data after finishing processing the initial data on it. However due to the remaining disk space of node i, the amount of data it can receive will not exceed R_i, therefore the function max is used. We let the basic unit of data transmission be a block, therefore the operators $\lceil\rceil$ and $\lfloor\rfloor$ are used. It is obvious that T_{tran} is determined by when the last node finishes its data transmission, therefore, we let all nodes finish their data transmission at the same time to minimize the total network bandwidth usage. In this situation,

$$T_{tran} = \frac{Q_i}{B_i} = \frac{Q_j}{B_j} \quad (5)$$

where symbol B_i stands for the bandwidth usage of node i. Then we obtain

$$B_i = \frac{Q_i}{T_{tran}} \quad (6)$$

To keep T_{tran} as small as possible, we should increase B_i. However, it is difficult to calculate B_i directly from the network. Therefore, we propose another way to obtain T_{tran}. Note that if there is a feasible T_{tran} that can satisfy the condition that T_{comp} is not bigger than T_{max}, then there must be a critical value $T^*_{tran}(0 \le T^*_{tran} < +\infty)$, such that

1. If $T_{tran} < T^*_{tran}$, there is no feasible solution which can satisfy $T_{comp} \le T_{max}$.
2. If $T_{tran} \ge T^*_{tran}$, there must be at least one feasible solution.

Let $G = (V, E)$ be the undirected graph of the cluster with compute node set V, arc set E. Each arc(u, v) stands for a network transmission channel between node u and node v. We also assume that P_i, C_i, M_i, and F_i of each node are known. Therefore, we can figure out the T^*_{tran} by binary search shown in Algorithm 1.

Algorithm 1 is similar to the classic binary search algorithm [7], but we still provide the pseudocode for intuition. Each time we obtain a possible T_{tran}, we check whether it is feasible by calling the function $Check(T_{tran})$. This function will be called $log\ T$ times, where T denotes the upper bound of T_{tran}. Details are presented in the next subsection.

Algorithm 1 Binary search to figure out T^*_{tran}

Require:
$G = (V, E), P_i, C_i, M_i, F_i, 1 \le i \le N$
Ensure:
T^*_{tran}

```
Left = 0,Right = INF
// Take a tolerance eps and stop when Right and Left differ by less than eps.
while Right − Left > eps do
    T_tran = (Right + Left)/2
    if Check(T_tran) == True then
        Right = T_tran
    else
        Left = T_tran
    end if
end while
T*_tran = (Right + Left)/2
return T*_tran
```

2.3 Establish flow network

In this subsection, we will discuss the method that we use to check whether a specific T_{tran} is feasible. We utilize the algorithm which is used to solve the maximum flow problem [28]. We need to build a flow network according to the cluster. For the sake of simplicity, we take two steps to build the flow network. Firstly, we build a flow network according to the Q_i of each node and the bandwidth between each two nodes, without considering the C_i. Secondly, we take C_i into consideration and establish the final flow network. We follow the steps below to build the first phase flow network:

Step 1 Let $G' = (V', E')$ be a graph with $V' = V$, $E' = \emptyset$. Each arc $(u, v) \in E'$ has an associated non-negative real valued capacity $C'(u, v)$, which means the maximal amount of data that can be transmitted between node u and node v.

Step 2 Add a source S and a sink T to V'.

Step 3 For each node i in V, we calculate Q_i using Eq. (4). If $Q_i > 0$, add a directed arc (S, i) to E', and let $C'(S, i) = Q_i$. If $Q_i < 0$, add a directed arc (i, T) to E', let $C'(i, T) = -Q_i$.

Step 4 For each arc (u, v)in E, add an undirected arc (u, v) to E', let $C'(u, v) = C'(v, u) = \lfloor b(u, v)T_{tran} \rfloor$ where $b(u, v)$ stands for the bandwidth between node u and node v.

The value of $C'(S, i)$ means the amount of data that node i should transmit, similarly, the value of $C'(i, T)$ stands for the amount of data that node i can receive. The value of $C'(u, v)$ stands for the maximum amount of data that can be transmitted between node u and node v during T_{tran}. Therefore, a feasible flow of G' stands for a feasible data transmission path. For example, suppose the cluster shown in Fig. 2 has the following properties:

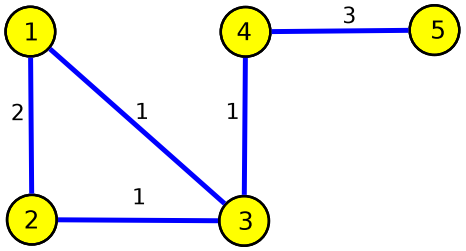

Fig. 2 A simple cluster

Table 2 Properties of the cluster

Property	Value
T_{max}	1.5
C_i	[1, 2, 1, 1, 1]
M_i	[3, 0, 2, 0, 0]
F_i	[False, False, False, False, False]
P_i	[1, 3, 0.5, 1, 1]
R_i	[1, 5, 2, 2, 2]

As shown in Table 2, C_i = [1, 2, 1, 1, 1] means that C_1 = 1,C_2 = 2,...,C_5 = 1. Other properties follow the same rules. We substitute P_i, M_i, F_i, R_i and T_{max} into Eq. (4), and obtain Q_i = [2, -4, 2, -1, -1]. Suppose we want to check whether T_{tran} = 2 is feasible, we firstly build the flow network following the steps above, the process is shown in Fig. 3.

Note that so far we do not consider the maximal transmission rate C_i of each node, to take it into consideration, we need another flow network based on Fig. 3. We divide each node $i (1 \leq i \leq N)$ into two nodes : node $i+$ and node $i-$, as shown in Fig. 4. Let $C''(u, v)$ stand for the capacity of arc (u, v). In a feasible flow, we have

$$\sum_{p=1}^{n} Fin\_p = C_i = \sum_{q=1}^{m} Fout\_q \tag{7}$$

and

$$C_i \leq C''(i+, i-) \tag{8}$$

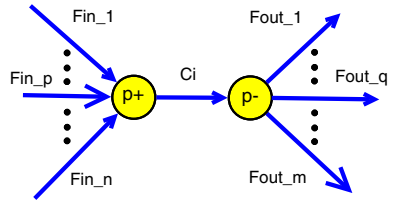

Fig. 4 Divide each node into two nodes

In this way, neither the amount of input flow nor output flow of node i will exceed $C''(i+, i-)$. The algorithm below illustrates the steps of building the flow network of the second phase. Figure 5 illustrates the final flow network after all above steps.

Step 1 Let $G'' = (V'', E'')$ be a directed graph with $V'' = \{S, T\}$, $E'' = \emptyset$. Each arc $(u, v) \in E''$ has an associated non-negative real valued capacity $C''(u, v)$.

Step 2 For each node i in V, add two nodes: node $i+$ and node $i-$ to V''.

Step 3 For each directed arc (S, i) in E', add a directed arc $(S, i+)$ to E'', let $C''(S, i+) = C'(S, i)$. Similarly, for each directed arc (i, T) in E', add a directed arc $(i-, T)$ to E'', let $C''(i-, T) = C'(i, T)$.

Step 4 For each undirected arc (u, v) in E', add two directed arcs $(u-, v+)$ and $(v-, u+)$ to E''. Let $C''(u-, v+) = C''(v-, u+) = C'(u, v)$.

Step 5 For each node i in S', add a directed arc $(i+, i-)$ with $C''(i+, i-) = \lfloor C_i T_{tran} \rfloor$ to E''.

After establishing the flow network, we obtain a maximum flow of it. The result is shown in Fig. 6. If the sum of flow to the sink T equals to the sum of all $Q_i (Q_i > 0, 1 \leq i \leq N)$, then the current T_{tran} is feasible. It is easy to obtain the real feasible flow from the flow shown in Fig. 6. We remove the source S and sink T, combine node $i+$ and node $i-$ into node i, and Fig. 7 illustrates the result. The number on each arc indicates the amount of data that are transmitted during T_{tran}. The flow size of arc $(S, i+)$ means the total amount of data that node i transmits to other nodes, the flow size of

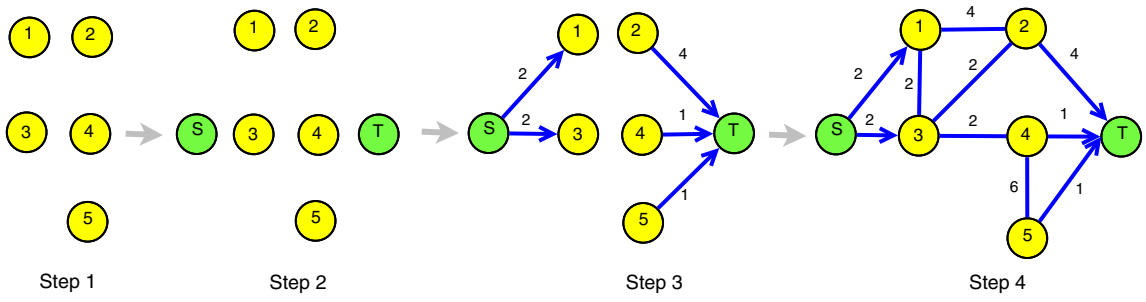

Fig. 3 Process of building flow network

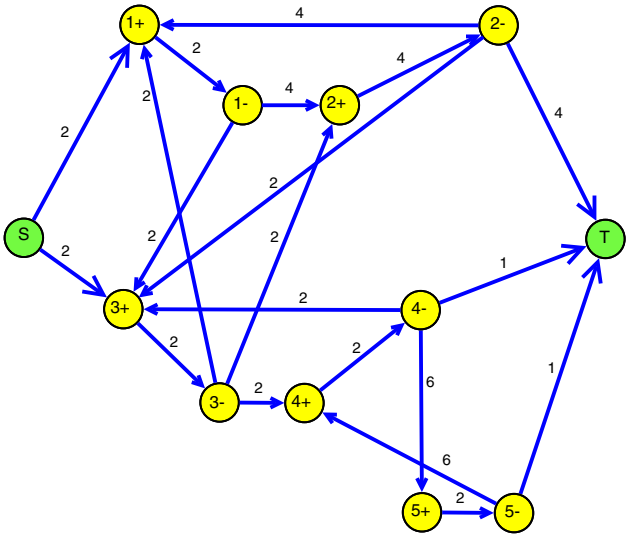

Fig. 5 Final flow network

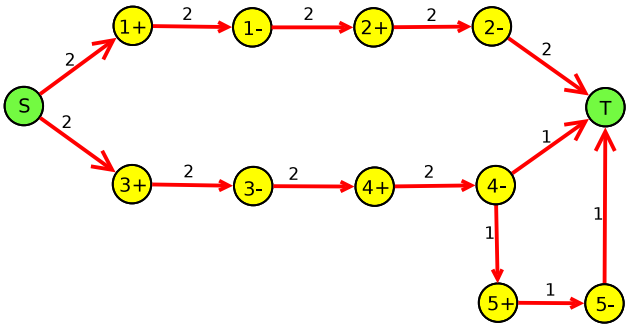

Fig. 6 A feasible flow when $T_{tran} = 2$

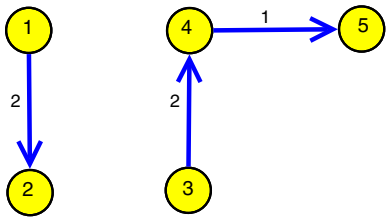

Fig. 7 The real feasible flow when $T_{tran} = 2$

arc $(i-, T)$ means total amount of data that node i receives from other nodes.

Up to this point, we have discussed the way to judge whether a specific T_{tran} is feasible, and figured out the T^*_{tran}. The function $Check(T_{tran})$ contains two main steps : establishing the flow network and calculating the maximum flow of it, and the latter will take much more time when N is large. Let M stand for the number of arcs in G. The process of building the network flow is the process of adding nodes and arcs to G'', which has $2N + 2$ nodes, and $2M + N + N$ arcs. Therefore the process of establishing the network flow

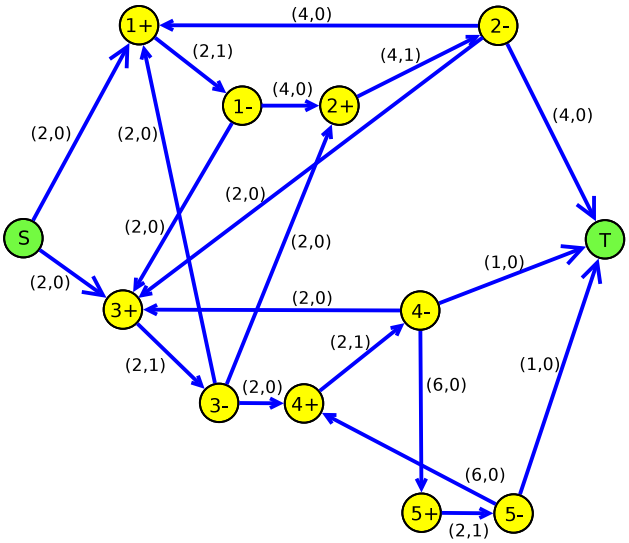

Fig. 8 The flow network with cost

takes $O(4N + 2M + 2)$ steps. In the next section we will discuss how to minimize the network bandwidth usage.

3 Optimizing

3.1 Minimize the occupied bandwidth

In Sect. 2 we discuss how to find out T^*_{tran} and a feasible flow, which can satisfy the condition $T_{comp} \leq T_{max}$. However the feasible flow is not optimal. Now our goal is to minimize the total occupied bandwidth in the data transmission phase. To achieve that goal, we let $G_c = (V_c, E_c)$ be a flow network with $V_c = V''$, $E_c = E''$. Each arc $(u, v) \in E_c$ has an associated real valued pair $(C_c(u, v), A(u, v))$. Let $C_c(u, v)$ equal to $C''(u, v)$ and let $A(u, v)$ be the unit cost of arc (u, v). The value of $A(i+, i-)(1 \leq i \leq N)$ is 1, while $A(u, v)$ equals to zero in the other circumstance. Figure 8 illustrates the flow network with cost of G'' shown in Fig. 5.

Let f_c denote a feasible flow of G_c, the total cost of f_c is

$$C(f_c) = \sum_{i \in V} f(i+, i-) \tag{9}$$

where $f(i+, i-)$ denotes the flow size between node $i+$ and $i-$. Note that $f(i+, i-)$ can also denote the bandwidth usage of node i, thus the total bandwidth usage equals to $C(f_c)$. Therefore, we calculate the minimum cost and maximum flow of G_c using any of these algorithms: Double Scaling: $O(nm(\log\log U)log(nC))$ [2], Jens Vygen & Orlin: $O(m \log m(m+n\log n))$ [31] etc.. Note that, the "minimum cost & maximum flow" algorithm can find a flow which is maximum, but has the lowest cost among the maximums.

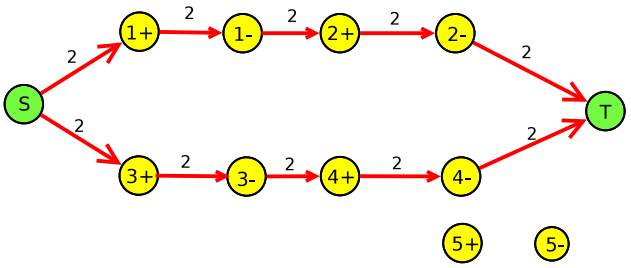

Fig. 9 A feasible minimum cost and maximum flow

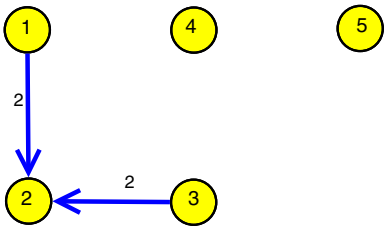

Fig. 10 The optimized flow network with a minimum number of nodes

One of the feasible flow is shown in Fig. 9. The number on each arc indicates the total amount of data that are transmitted during T_{tran}.

3.2 Minimize the number of nodes

We can calculate the amount of final data (D_i) of each node. For example, we can obtain D_3 after data transmission: $D_3 = M_3 - 2 = 2 - 2 = 0$. We can obtain $D_i = [1, 2, 0, 2, 0]$. After calculating the amount of final data of each node, we find that only node 1, node 2, and node 4 participate in the data computation phase. We discover that if node 3 transmits all its data to node 2 instead of node 4, the result ($D_i = [1, 4, 0, 0, 0]$) will also be a feasible flow, however, the number of nodes which participate in the data computation phase is less than that in Fig. 9. We divide all the nodes in f_c into three categories:

Type 1 Nodes that do not participate in data computation phase ($D_i = 0$).
Type 2 Nodes that participate in data computation because they receive data from other nodes ($D_i > 0, M_i = 0$).
Type 3 Nodes that must participate in data computation ($D_i > 0, M_i > 0$).

We use a greedy algorithm to change nodes of type 2 into type 1. Steps below illustrate how to reduce the number of nodes:

Step 1 For each node i in f_c, if node i is of type 1 and $(i-, T) \in E_c$, delete arc $(i-, T)$ in G_c.
Step 2 Sort the nodes of type 2 in f_c by D_i from smallest to largest, let N_{sorted} denote the the list of sorted nodes.
Step 3 For each node i in N_{sorted}, delete arc $(i-, T)$ in G_c, then try to find a feasible flow of minimum cost & maximum flow of G_c. If there is no a feasible flow or the cost of this feasible flow is larger than before, then recover the arc that has just been deleted.

Note that in the flow network, the total input flow size of a node must equal to the total output flow size, therefore arc $(i-, T)$ must exist so that node i of type 2 ($D_i > 0, M_i = 0$) can receive data from other nodes. Figure 10 shows an optimized flow network of Fig. 9. In the optimized flow network, only two nodes (node 1 and node 2) participate in the computation phase.

4 Replication

In this section, we describe how our algorithm works when the data are replicated. Before the discussion of replication, we want to introduce the concept of copyset [6]. A copyset is a set that stores all of the copies of a specified data, which can be a file or a chunk. Asaf Cidon et al. pointed out that a large number of distinct copysets will increase the probability of losing data under a massive correlated failure. We make our algorithm available under the situation of replication when the number of copysets is not too large. Our solution is adding virtual nodes, each of which stands for a copyset, into our initial network topology. Therefore if the number of copysets is too large, the total number of nodes will be too large. According to the study of Asaf Cidon et al., a well designed storage system should have a small number of copysets, in which situation our solution will work fine. Specifically, when we consider the situation of replication, we do the following things to the initial network topology $G = (V, E)$:

Step 1 Add N_c virtual nodes to V, where N_c is the number of copysets. Each virtual node represents a copyset.
Step 2 Link each virtual node with its corresponding real nodes, let the bandwidth between the virtual node and its corresponding real nodes be infinity.

Now we get a new network topology , when a task begins, we do the following things:

Step 1 Let the busy flag of each virtual node be busy. Because the virtual nodes do not have any compute ability.
Step 2 For each virtual node i, let M_i be the total amount of initial data located in its corresponding copyset, the

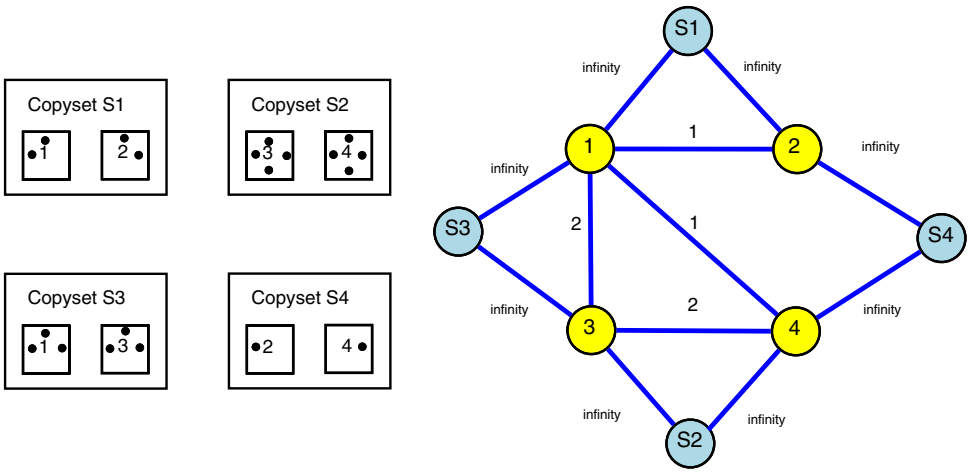

Fig. 11 The copysets and the initial network with virtual nodes

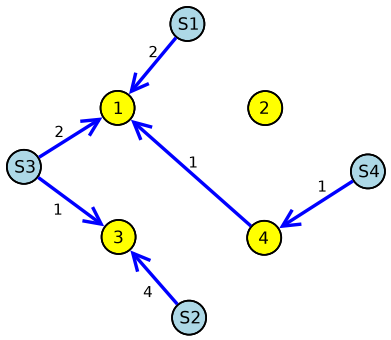

Fig. 12 A feasible flow in replication situation

same data will be counted only once. Let M_j be zero where node j is a real node.

In this way, a virtual node has to transmit all its data to its corresponding real nodes. Because that each data has several replicas, however only one of them should be processed (calculated or transmitted to other nodes) during one task. Therefore we need a scheme of data distribution to decide on which node a specified data should be processed. The way a virtual node transmits its data represents the scheme of data distribution we need. For a specific copyset, if the virtual node i transmits k data to one of its corresponding real node j, it means that the amount of data which are processed in node j is k.

The remaining steps are the same with what we mention in Sects. 2 and 3. Here is an example of how we modify the initial network topology. Suppose we have four real nodes, node 1 $\sim$ node 4, four copysets, copyset $S1 \sim S4$. Figure 11 shows the copysets and the initial network with virtual nodes. The black spots in each node represent the data, for example the amount of data of copyset $S1$ is 2. A possible feasible flow in this situation is shown in Fig. 12.

The data transmission of the virtual node means the data distribution of the real nodes in the same copyset in the data computation phase. For example, as shown in Fig. 12, the virtual node $S3$ transmits all its data to node 1 and node 3, it means that, for copyset $S3$, the amount of data that are processed in node 1 is 2, while the amount of data that are process in node 3 is 1.

5 Simulation

A series of experiments are conducted to evaluate the effect of our algorithm as well as our optimization which is discussed in Sect. 3. We compare our algorithm to the algorithm which is used in Hadoop to reduce bandwidth usage. Before optimization, the algorithm focuses on solving the data skew problem, and make the T_{comp} not larger than T_{max}, but it will occupy too much bandwidth. To see how the occupied bandwidth and the number of nodes which take part in computation phase decrease after the optimization, we pick three typical variables: T_{max}, total amount of data, total number of nodes. Then we compare the optimized algorithm to the algorithm which is used to schedule the map tasks in Hadoop[1] to reduce bandwidth usage. The nodes are arranged in a tree network, because tree topology is most typical in real world. In every experiment, we randomly let half of the nodes be busy. Other variables are generated randomly, such as P_i, C_i, M_i, and R_i. Each experiment is repeated over 100 times and the result shows the average situation. The network topology is shown in Fig. 13. All the nodes are arranged in the leafs of the tree, and each inner node denotes a switch, which can connect to up 32 other switches or nodes. We build the tree from bottom to top: every 32 nodes are connected to a switch, and every 32 switches are connected to an upper layer switch. For example, if we have 4096 nodes, then the tree will have 4 layers: $4096 = 32 \times 32 \times 4 \times 1$.

5.1 T_{max}

In this experiment, we aim at finding the effect of T_{max}. The network topology is shown in Fig. 13. Table 3 summarizes

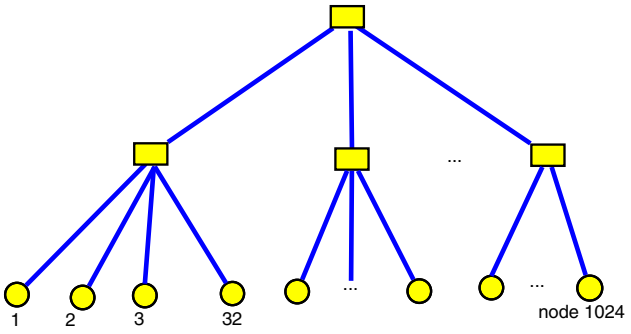

Fig. 13 The network topology in the experiments

Table 3 Range of the variables

Variable	Range
P_i	(0, 128)
R_i	(0, 512)
C_i	(0, 1024)
M_i	(0, 1024)
$\sum_{i=1}^{N} M_i$	100,000
F_i	Half of the nodes will be busy.
N	1024
T^*_{max}	$(0, +\infty)$

* The symbol * means that the variable is a controlled argument in this experiment

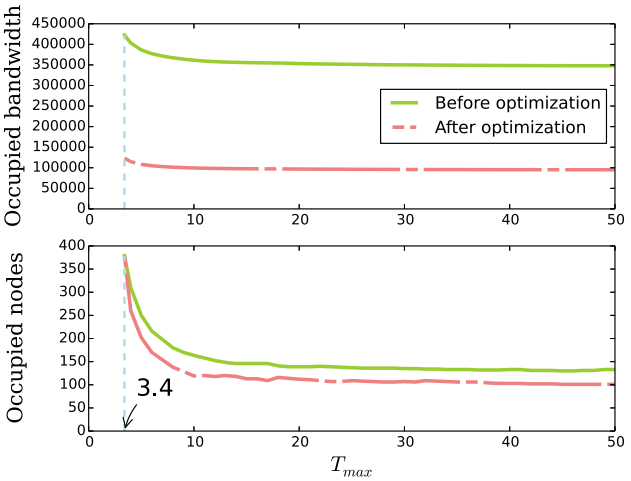

Fig. 14 The occupied bandwidth and nodes of different T_{max}

the variables that we use in this experiment. The results are shown in Fig. 14.

As we saw from the scenario of Fig. 14, the occupied bandwidth and number of nodes that take part in computation can be decreased by about 73 and 33 % respectively when T_{max} grows up. There is no feasible flow when T_{max} is too small (less than 3.4 in this experiment).

Table 4 Range of the variables

Variable	Range
P_i	(0, 128)
R_i	(0, 512)
C_i	(0, 1024)
M_i	(0, 1024)
$\sum_{i=1}^{N} M^*_i$	(2500, 2500 * 50)
F_i	Half of the nodes will be busy.
N	1024
T_{max}	5

* The symbol * means that the variable is a controlled argument in this experiment

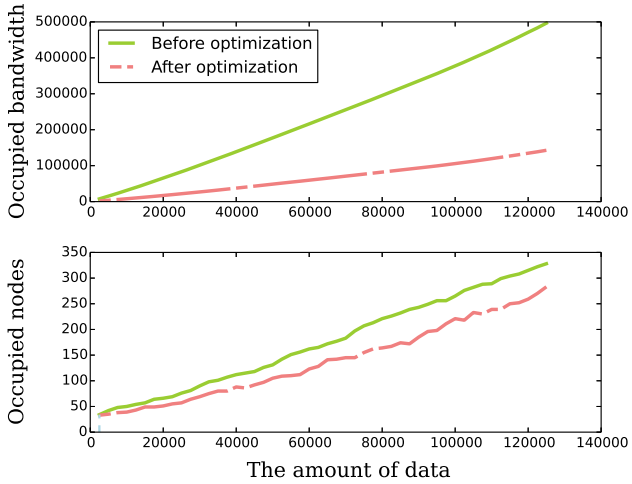

Fig. 15 The occupied bandwidth and nodes of different amount of data

5.2 The amount of data

In the second experiment, we study how the amount of data can effect the result of optimization. Table 4 summarizes the variables that we use in this experiment. We conduct a series of tests, the initial amount of data is 2500. We increase the amount of data by 2500 in each test, based on the data distribution in previous test. In this way we keep the amount of data changing while not influencing too much on the data distribution. The results are shown in Fig. 15. The occupied bandwidth and number of nodes that take part in computation can be decreased by about 71 and 14 % respectively when the amount of data grows up.

5.3 The number of nodes

In the third experiment, we study how the number of nodes can effect the result of optimization. Table 5 summarizes the variables that we use in this experiment. The results are

Table 5 Range of the variables

Variable	Range
P_i	(0, 128)
R_i	(0, 512)
C_i	(0, 1024)
M_i	(0, 1024)
$\sum_{i=1}^{N} M_i$	100,000
F_i	Half of the nodes will be busy.
N^*	(256, 10,000)
T_{max}	10

* The symbol * means that the variable is a controlled argument in this experiment

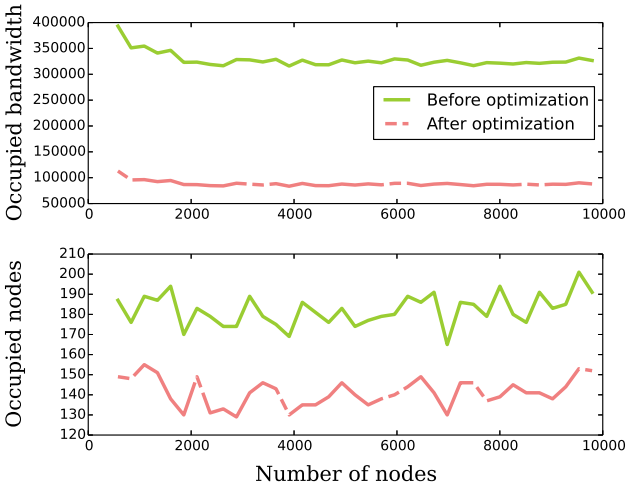

Fig. 16 The occupied bandwidth and nodes of different number of nodes

shown in Fig. 16. The occupied bandwidth can be decreased by about 72 % when the number of nodes grows up. However, the number of nodes that take part in computation is volatile. This is because when we change the number of nodes, the network topology changes too, which results in the change of data distribution.

5.4 Comparison to another bandwidth saving algorithm

In the fourth experiment, we compare our optimized algorithm to the algorithm which is used to schedule the tasks in Hadoop to reduce bandwidth usage. The strategy of moving computation to the data, instead of moving the data to the computation allows Hadoop to achieve high data locality which in turn results in less bandwidth usage. In this experiment, we simulate the default hadoop scheduler called JobQueueTaskScheduler which can be found in the source code of Hadoop, and show the results of bandwidth usage in both situations. When simulating the task in Hadoop, each

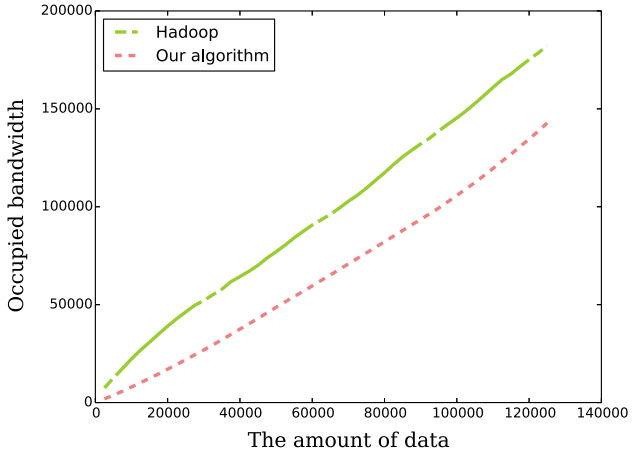

Fig. 17 The occupied bandwidth in our algorithm and Hadoop

block of data are processed in a map task, there are no reduce tasks in this experiment. The variables that we use in this experiment are the same as those in second experiment. The results are shown in Fig. 17.

5.5 Discussion

A series of experiments are conducted to evaluate the effect of our optimization. To see how the occupied bandwidth and the number of nodes which take part in computation phase decrease after the optimization, we pick three typical variables: T_{max}, total amount of data, total number of nodes. The experimental results show that our algorithm can effectively reduce the occupied bandwidth (about 70 %). The second and third experiments prove that our algorithm can work fine under a large scale of system. Moreover, the number of nodes that participate in data computation phase can also decrease, in this way, part of the computational resources can be saved. We think the number of nodes which take part in computation phase can be further reduced. In the fouth experiment, we compare our algorithm to the default algorithm which is used in Hadoop to reduce bandwidth usage, and find that our algorithm can still have a better effect on reducing bandwidth usage.

6 Related work

Data skew has been widely studied in the literature, and a large number of skew aware load balancing algorithms have been developed, some of which are quite successful. Abdelsalam (Sumi) Helal et al. introduced a dynamic and transactional re-allocation scheme based on the work on disk cooling in shared memory architecture by Scheuermann et al. The proposed scheme detects access skew as it occurs and re-allocates data partitions to underloaded processing

Table 6 Polynomial algorithms for the max flow problem

Year	Due to	Running time
1956	Ford & Fulkerson	$O(nmU)$
1977	Malhotra, Kumar & Maheshwari	$O(n^3)$
1989	Ahuja, Orlin & Tarjan	$O(nm\,log(n\sqrt{U}/(m+2)))$
1996	Cheriyan, Hagerup & Mehlhorn	$O(n^3/log\,n)$
2012	Orlin	$O(nm)$
2012	Orlin	$O(n^2/log\,n)$ if $m = O(n)$

elements on the fly [15]. Hongjun Lu et al. presented a fully dynamic partitioning approach that could effectively distribute the workload among both intra- and inter-processing nodes without priori knowledge of data distribution [24]. In recent years, with the development of MapReduce, some algorithms that focus on solving the data skew in MapReduce are presented, such as [14,17,29]. However, none of these algorithms considers the network resources usage. Resources of a distributed system can be divided into computational resources and network resources. Due to the rapid development of the microprocessor, the effects of the computation delay on the overall performance of the system are becoming less significant. On the other hand, the communication delay still has a major effect on the system performance and can be a major problem especially in a system with many nodes and a high communication frequency. A lot of efforts have been made to reduce the network resources usage in distributed systems. For example, Min Li et al. proposed a cloud platform called CAM that provides an innovative resource scheduler particularly designed for hosting MapReduce applications in the cloud, their system could reduce network traffic and average MapReduce job execution time by a factor of 3 and 8.6, respectively [23]. Dzmitry Kliazovich et al. presented a scheduling approach that combines energy efficiency and network awareness, named DENS [21]. The DENS methodology balances the energy consumption of a data center, individual job performance, and traffic demands. The proposed approach optimizes the trade-off between job consolidation (to minimize the amount of computing servers) and distribution of traffic patterns (to avoid hotspots in the data center network). Unfortunately, none of these algorithms considers the trade-off between the computational resources and the network resources.

Some researches have been made to use the maximum flow algorithm to reduce network resources in the distributed systems. The algorithm called Balance-Reduce(BAR) [20] was proposed to reduce the job completion time gradually by tuning the initial task allocation. Differ from our algorithm, it only aims at reducing the completion time of tasks integrally. Instead that our algorithm focus on optimizing both the completion time and the bandwidth of each single task to achieve better effect. Both of BAR and our work consider the network state and cluster workload in the algorithms. The authors of [27] proposed a low cost network coding algorithm based on the maximum flow algorithm and key link. Differently from our model, it focuses on data transmission while our model contains both data transmission and data computation. Another data transmission algorithm to deal with the load rebalancing problem in distributed file systems in clouds has been presented in [16]. Compared with our work, [16] requires data transmission to reduce the demanded network traffic caused by rebalancing the loads of nodes as much as possible, while our algorithm aims at minimize network bandwidth usage and data transmission time.

The maximum flow problem was first formulated in 1954 by T. E. Harris as a simplified model of Soviet railway traffic flow in 1955 [28]. Lester R. Ford, Jr. and Delbert R. Fulkerson created the first known algorithm, the Ford-Fulkerson algorithm [11,12]. Over the years, various improved solutions to the maximum flow problem were discovered, notably the shortest augmenting path algorithm of Edmonds and Karp [10] and independently Dinitz; the blocking flow algorithm of Dinitz [9]; the push-relabel algorithm of Goldberg and Tarjan [4]; and the binary blocking flow algorithm of Goldberg and Rao [13]. The electrical flow algorithm of Christiano et al. [5], and Spielman finds an approximately optimal maximum flow but only works in undirected graphs [30]. The following table lists the time complexities of different algorithms for solving the maximum flow problem. Table 6 summarizes these developments.

7 Conclusion and future work

In this paper, we introduce the distributed two-phase model (DTPM), and propose an algorithm to minimize the data transmission time and the bandwidth usage in this model, while a given upper bound of relative computation time is satisfied. The algorithm we use to establish the flow network takes $O(4N + 2M + 2)$. Besides, we introduce the way to reduce the number of nodes that participate in the data computation phase. The results show that the occupied bandwidth can be reduced effectively (about 70 %) in the situation of large-scale data sets and large number of nodes. We describe

how our algorithm can work in replication situation while the number of copysets is not large. Although priority is not our main subject, we can support different priorities of different user, without changing our algorithm. We can achieve priority by adding a logical layer. For example, a task of an user with high priority can be launched earlier. We can also limit the range of T_{comp} of users with different priorities. We also plan to consider several high-level objectives, such as intelligent scheduler based on genetic algorithm and energy saving.

Acknowledgments This work was supported by the National 863 Project (2011AA040101) and was jointly funded by the Beijing municipal Education Commission of the Scientic Research.

References

1. Apache hadoop. http://www.hadoop.apache.org
2. Ahuja, R.K., Goldberg, A.V., Orlin, J.B., Tarjan, R.E.: Finding minimum-cost flows by double scaling. Math. Program. **53**(1–3), 243–266 (1992)
3. Buyya, R.: Parmon: a portable and scalable monitoring system for clusters. Software **30**(7), 723–740 (2000)
4. Cherkassky, B.V., Goldberg, A.V.: On implementing the pushrelabel method for the maximum flow problem. Algorithmica **19**(4), 390–410 (1997)
5. Christiano, P., Kelner, J.A., Madry, A., Spielman, D.A., Teng, S.H.: Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing, pp. 273–282. ACM Press, San Jose (2011)
6. Cidon, A., Rumble, S., Stutsman, R., Katti, S., Ousterhout, J., Rosenblum, M.: Copysets: reducing the frequency of data loss in cloud storage. In: Presented as part of the 2013 USENIX Annual Technical Conference, pp. 37–48. USENIX (2013)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.L., et al.: Introduction to Algorithms. MIT Press, Cambridge (2001)
8. Dean, J., Ghemawat, S.: Mapreduce: a flexible data processing tool. Commun. ACM **53**(1), 72–77 (2010). doi:10.1145/1629175.1629198
9. Dinic, E.: Algorithm for solution of a problem of maximum flow in a network with power estimation. Soviet Math. Doll. **11**(5), 1277–1280, (1970) (English translation by RF. Rinehart)
10. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM (JACM) **19**(2), 248–264 (1972)
11. Ford, D., Fulkerson, D.R.: Flows in Networks. Princeton University Press, Princeton (2010)
12. Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Can. J. Math. **8**(3), 399–404 (1956)
13. Goldberg, A.V., Rao, S.: Beyond the flow decomposition barrier. J. ACM (JACM) **45**(5), 783–797 (1998)
14. Gufler, B., Augsten, N., Reiser, A., Kemper, A.: Handling Data Skew in Mapreduce, pp. 574–583. Eindhoven University of Technology, Noordwijkerhout (2011)
15. Helal, A.S., Yuan, D., Hesham, E.R.: Dynamic data reallocation for skew management in shared-nothing parallel databases. Distrib. Parallel Databases **5**(3), 271–288 (1997)
16. Hsiao, H.C., Chung, H.Y., Shen, H., Chao, Y.C.: Load rebalancing for distributed file systems in clouds. IEEE Trans. Parallel Distrib. Syst. **24**(5), 951–962 (2013). doi:10.1109/TPDS.2012.196
17. Ibrahim, S., Jin, H., Lu, L., He, B., Antoniu, G., Wu, S.: Handling partitioning skew in mapreduce using leen. Peer-to-Peer Netw. Appl. **6**(4), 409–424 (2013)
18. Imamagic, E., Dobrenic, D.: Grid infrastructure monitoring system based on nagios. In: Proceedings of the 2007 Workshop on Grid Monitoring, pp. 23–28. ACM Press, New York (2007)
19. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks. SIGOPS Oper. Syst. Rev. **41**(3), 59–72 (2007). doi:10.1145/1272998.1273005
20. Jin, J., Luo, J., Song, A., Dong, F., Xiong, R.: Bar: an efficient data locality driven task scheduling algorithm for cloud computing. In: Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on, pp. 295–304. IEEE Press, New York (2011)
21. Kliazovich, D., Bouvry, P., Khan, S.U.: Dens: data center energy-efficient network-aware scheduling. Clust. Comput. **16**(1), 65–75 (2013)
22. Kwon, Y., Balazinska, M., Howe, B., Rolia, J.: Skewtune: mitigating skew in mapreduce applications. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 25–36. ACM Press, New York (2012)
23. Li, M., Subhraveti, D., Butt, A.R., Khasymski, A., Sarkar, P.: Cam: a topology aware minimum cost flow based resource manager for mapreduce applications in the cloud. In: Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, pp. 211–222. ACM Press, Hoboken (2012)
24. Lu, H., Yu, J.X., Feng, L., Li, Z.: Fully dynamic partitioning: handling data skew in parallel data cube computation. Distrib. Parallel Databases **13**(2), 181–202 (2003)
25. Märtens, H.: A classification of skew effects in parallel database systems. In: Euro-Par 2001 Parallel Processing, pp. 291–300. Springer, New York (2001)
26. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Comput. **30**(7), 817–840 (2004)
27. Run-liu, W., Yun-hui, Y.: Low cost network coding algorithm for data distribution network. In: Proceedings of 8th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), pp. 1–4 (2012). doi:10.1109/WiCOM.2012.6478566
28. Schrijver, A.: On the history of combinatorial optimization (till 1960). Handbook of Discrete Optimization pp. 1–68 (2005)
29. Slagter, K., Hsu, C.H., Chung, Y.C., Yi, G.: Smartjoin: a network-aware multiway join for mapreduce. Clust. Comput. **17**, 1–13 (2014)
30. Spielman, D.A., Teng, S.H.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: Proceedings of the 36h Annual ACM Symposium on Theory of Computing, pp. 81–90. ACM Press, New York (2004)
31. Vygen, J.: On dual minimum cost flow algorithms. Math. Methods Oper. Res. **56**(1), 101–126 (2002)
32. Yook, J., Tilbury, D.: Performance evaluation of distributed control systems with reduced communication. Ann Arbor **1001**, 48,109 (2001)
33. Yook, J.K., Tilbury, D.M., Soparkar, N.R.: Trading computation for bandwidth: reducing communication in distributed control systems using state estimators. IEEE Trans. Control Syst. Technol. **10**(4), 503–518 (2002)

Xiaolu Zhang is currently a graduate student and working towards his M.E. degree at University of Science and Technology Beijing, China. His major is Computer Science and Technology. His current research interest includes distributed storage and algorithm design.

Jiafu Jiang is currently a graduate student and working towards his M.E. degree at University of Science and Technology Beijing, China. His major is Computer Science and Technology. His current research interest includes cloud computing and distributed computing.

Xiaotong Zhang received the Ph.D. degrees from University of Science and Technology Beijing, in 1997, and 2000, respectively. He was an Assistant Professor, an Associated Professor and Professor in the Department of Computer Science and Technology, University of Science and Technology Beijing, from 2000 to 2009. He was the visiting scholar from 2010 to 2011 in LONG Lab of Department of Computer Science and Engineering of Lehigh University. Now he is Vice President of the Department of Computer Science and Technology, University of Science and Technology Beijing. His industry experience includes affiliation with Beijing BM Electronics High-Technology Co., Ltd. from 2002 to 2003, where he worked on digital video broadcasting communication systems and IC design, His industrial cooperation experience includes BLX IC Design Co., Ltd, North Communications Corporation of PetroChina, and Huawei Technologies Co., Ltd. etc. His research includes work in quality of wireless channels and networks, wireless sensor networks, networks management, cross-layer design and resource allocation of broadband and wireless network, signal processing of communication, computer architecture, the technology of big data, cloud computing, distributed system.

Xuan Wang is currently studying at University of Science and Technology Beijing for a doctorate. Her major is Computer Science and Technology. Her research includes work in load balancing of disturbed system, distributed storage, cloud computing, algorithm design and distributed computing.