# Virtual Machine Placement to Minimize Data Transmission Latency in MapReduce

Jie Wei, Shangguang Wang, Lingyan Zhang, Ao Zhou, Qibo Sun and Fangchun Yang
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Haidian, Beijing, 100876, China
{wjwb; sgwang; zhanglingyan; aozhou; qbsun; fcyang}@bupt.edu.cn

## Abstract

Many factors affect the time cost of Cloud computing tasks. One of the most serious factors is data transmission latency, which reduces the efficiency of Cloud computing. Many notable schemes that have been proposed to overcome this factor ignore the communication cost among virtual machines (VMs) in the MapReduce environment. In this paper, we propose a VM placement approach to reduce data transmission latency by focusing on the communication cost among VMs. In this approach, we first propose two VM placement optimization algorithms to minimize the total data transmission latency and the maximum data transmission latency in the MapReduce environment. Then, we use the algorithms to place VMs for Map and Reduce phase. Finally, we analyze the time complexity for our approach. We implement our approach by simulation. The simulation results show that our approach reduces the average data transmission latency by 26.3% compared with other approaches.

**Keywords:** virtual machine placement; data transmission latency; MapReduce; data node

## 1 Introduction

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster [1]. It is based on the common use of Map and Reduce operations in functional programming. The model undertakes efficient parallel computing through a large number of data nodes and computation nodes for data intensive Cloud application. The total completion time is an important performance metric for Cloud application. There are many influencing factors for the completion time, such as communication bottlenecks, the load of servers, etc. However, the main impact factor is the transmission latency between data nodes and computation nodes. As the time cost of each (MapReduce) task can affect the economic benefits of cloud service providers, it is important and needs to be optimized. Meanwhile, the time cost is mainly determined by the data transmission latency between data nodes and VMs. Hence, reducing the data transmission latency is a primary problem for Cloud service providers [2, 3].

The data transmission latency of a given task is the time that data need for transmission from the relevant data nodes to computation nodes. The placement of the VMs exerts a tremendous influence on data transmission latency. A bad placement may lead to quite large transmission latency. The basic VMs placement schedule has two aspects. One is placing the VMs to the data nodes in which the data is stored, and the other is moving the data to computation nodes in which the VMs are located. However, neither of them is always feasible. As it is difficult to store data locally in most cases, we often encounter large data transmission latency. Some notable schemes have been proposed to minimize the latency by appropriate VMs placement ([4], [5], [6], [7]). These approaches generally choose some VM cliques, and then assign the chosen cliques to data nodes. Nevertheless, there are mainly four shortcomings in those approaches.

1) The previous approaches minimized transmission latency under the case that communication exists among each VMs. However, the latency among VMs only exists between the Map VMs and the Reduce VMs, that is to say, any two VMs in the same layer(Map or Reduce) do not need to communicate with each other for data exchanging (MapReduce contains Map layer and Reduce layer). Only the VM in the Map layer can accept data from the data node. VMs in the Reduce layer cannot receive data directly from the data node, but can accept data from Map layer VMs. Thus VMs which belong to the same layer do not have communication with each other at all. Data exchanging only occurs between VMs belonging to different layers. So, we just need to consider the latency between VMs which belong to different layers when we optimize the data transmission latency.

2) Existing VM placement approaches often choose the intensive VMs which are far from data nodes rather than the sparse VMs which are around data nodes. When the distribution of VMs is homogeneous, these existing approaches are effective. However, VMs distribution is often heterogeneous in practical applications. Because the previous approaches give priority to transmission latency among VMs (Map or Reduce) rather than latency between data nodes and Map VMs, these approaches will choose the more intensive VMs as Map VMs instead of the sparse VMs. When plenty of intensive VMs are far from data nodes and a small number of sparse VMs are near to data nodes, choosing the further and intensive VMs may significantly increase the data transmission latency. This will become more serious in the case that the latency

between data nodes and VMs is much bigger than the latency among VMs.

3) The time complexity (we analysis time complexity in the Appendix A) of existing VM placement approaches is very high. If the number of available VMs is *n*, then the approaches will create *n* cliques (each VM is the center of its clique). To find the total shortest data transmission latency between data nodes and VMs, the placement algorithm must be repeated *n* times. When *n* is very large, the time complexity of previous approaches becomes extremely high.

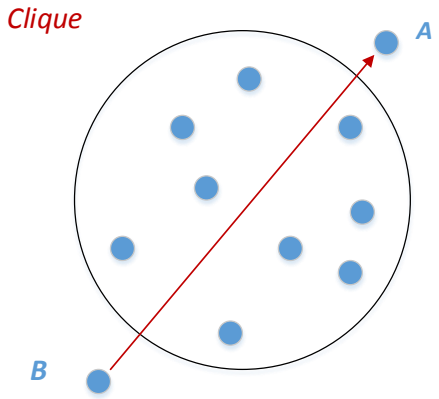


Figure.1 Both VM *A* and *B* satisfy the constraint of clique selection. However, only one of them will be added into the clique, since the latency between them exceeds the threshold.

4) The previous approaches are not totally correct and appropriate. When the radius of the clique is small, the approaches get many repeated cliques and the VMs in clique may be not sufficient for data nodes, which mean a low efficient work. When the radius of the cliques is too large that contains massive VMs, it means that the radius is out of action. In addition, it can encounter an unequal case. As shown in Figure 1, the small circles represent computation nodes (VMs), and the small circles in big round wire represent the VMs which have been added to clique. Now we judge if *A* and *B* should be added to clique. We assume that the latency between *A* and any VM in circle is below the clique selection threshold, and it is the same to *B*. It looks like both *A* and *B* should be added to clique. However, the latency between *A* and *B* is beyond the threshold, which means *A* and *B* cannot be added to the clique together and only one of them can be added to the clique. If choose *A* to be added, it is not fair for *B*, and vice versa. The previous approaches cannot judge which one should be added to clique.

To overcome these shortcomings, we propose a novel VM placement optimization approach to minimize the data transmission latency. We first propose two VM placement optimization algorithms to minimize the total data transmission latency and the maximum data transmission latency. The first optimization is modeled as a linear sum assignment problem (LSAP), and the second is formulated as a linear bottleneck assignment problem (LBAP). Then we use the algorithms to place VMs for Map and Reduce phase. The Reduce phase is based on the Map phase and the inter-VM latency. At last, we analyze the time complexity of our approach and the previous approaches. Compared to previous work, we decrease the data transmission latency between data node and its assigned Map VM enormously, and place VMs for Reduce phase in a fine grain level in heterogeneous environment. We also decrease the time complexity of VM placement approach. We evaluate our approach by massive simulation, and compare our approach with previous approaches in terms of the total data transmission latency, maximum data transmission latency, and time complexity. The experimental results show that our approach is superior to previous approaches.

The rest of this paper is organized as follows. First, in Section 2, we discuss some related work. We then present our novel VM placement optimization approach in Section 3. Section 4 describes the simulation results and studies the impact of various parameters. Finally, we conclude the paper in Section 5.

## 2 Related Work

Many researchers have proposed approaches that aim to minimize data transmission latency. Because of space constraints, we only review some notable approaches.

1) Minimization of time cost. The Cloud workflow is minimized in [8, 9], Pandey et al. [9] proposed an online linear programming model to minimize the data retrieval and time cost of data-intensive workflows in clouds. The model retrieves data from the storage of the cloud. The time cost of communication is proportional to the data volume. Based on the storage and computing resources of the cloud, the application is modeled as a workflow. Compared with Amazon CloudFront's "nearest" single data source selection, this model reduces the time cost of multiple executions by 75%. Feng et al. [10] presented an online optimization problem that minimizes the operational time cost using a store-and-forward procedure at intermediate nodes on the inter-datacenter traffic. By restricting data transmission to a time-slotted model, the problem was formulated on a time-expanded graph and solved by convex optimization solvers. As we all know, minimizing the time cost equals to maximizing the economic profits. A dynamic virtual resource renting approach has been introduced that adopts outlier detection to filter extreme prices [11, 12]. This method utilizes a weak equilibrium operator for the virtual resources, and a novel rental decision-making algorithm to select the most profitable resource.

2) Minimization of data latency in Hadoop [13, 14]. MapReduce processes many tasks in parallel. Thus, the completion time of a job is determined by the last finished task. Hadoop is an open-source implementation of MapReduce whose scheduler suffers from performance degradation in heterogeneous environments [15]. Zaharia et al. [16] designed a new scheduling algorithm to minimize the data transmission latency, and this improves Hadoop's response time enormously. Lin et al. [17] showed that a distributed cache can be adapted to Hadoop, and provide a feasible, scalable, and general-purpose solution. A system called Mantri has been developed that

uses a cause-and-resource-aware technique to monitor tasks and cull outliers [18]. This system considers a judicious arrangement with network bottlenecks, and uses a greedy algorithm to arrange Hadoop tasks and minimize data latency. Mantri decreases job completion times to 32%.

3) Minimization of data transmission by VM placement[19, 20]. Isard et al. [21] introduced a new, graph-based VM placement for the real-time distributed scheduling of jobs with fine-grained resources. As fairness and locality generally conflict, this scheme delays a VM's execution until the resource is available, which increases the probability of data local access? The proposed VM placement is assumed to operate on a fine-grained timescale, with fairness and locality reflected by the edge weights in the graph model. The method is then transformed into a min-cost flow problem. Zaharia et al. [22] presented a simple VM placement algorithm called delay scheduling to solve the conflict between fairness and locality. When fairness prevents a local VM task from being launched, other VM tasks are launched instead. Greenberg et al. [23] shows that it is important to retain a balance between fairness and locality. Kuo et al. [24] first propose a 3-approximation algorithm for minimizing the maximum data transmission latency. Subsequently, they close the gap by proposing a 2-approximation algorithm, which is an optimal approximation algorithm for resolving the problem in the price of higher time complexity. However, their optimization objective isn't totally same with ours. Alicherry and Lakshman [4] introduced an optimization approach based on VM placement. They considered the VM placement with and without inter-VM constraint. The established model assumes that the relationship among VMs satisfies the triangle inequality. This means that communication occurs between any two VMs. However, this is not suitable for MapReduce because latency only exists between VMs in different layers. The VM of Map can communicate with VM of Reduce, but it is impossible when the VMs come from the same layer. The Hungarian algorithm [25] is used to solve the assignment problem. When intensive distribution of VMs is far from the data nodes and sparse distribution of VMs is closer, this approach always chooses the intensively distributed VMs instead of the sparsely distributed VMs, which increases the data transmission latency significantly. When the inter-VM constraint is added to the problem, it becomes NP-hard. This case can be solved heuristically, but the characteristics of the heuristic algorithm mean there is a considerable amount of redundant time.

To the best of our knowledge, no previous method has considered the real relationship among the VMs in MapReduce. The latency only exists between VMs in Map layer and Reduce layer. Moreover, previous VM placement techniques often neglect sparse VMs that are close to the data nodes, and have very high time complexities. Hence, the data transmission latency has not been efficiently optimized.

# 3 Proposed Approach

In this section, we introduce an effective approach for the placement of VMs. First, we propose two optimization algorithms to minimize the total data transmission latency and the maximum data transmission latency. Then, we propose an approach to place VMs for Map and Reduce layer. We analyze the time complexity of our approach and the previous approach in the appendix. The notation used in this section is defined in Table 1.

Table 1 NOTATIONS

| Symbol | Meaning |
|---|---|
| $C_M$ | Placed Map VMs |
| $C_R$ | Placed Reduce VMs |
| $m$ | Cardinality of the data nodes and VMs of Map |
| $n$ | Cardinality of computation nodes |
| $k$ | Cardinality of Reduce |
| $D_i$ | Data nodes |
| $V$ | $V$ is the left vertex set of $G = (V \cup W; E)$; it represents data nodes |
| $W$ | W is the right vertex set of $G = (V \cup W; E)$; it represents computation nodes |
| $E$ | $E$ is the edge set between the two vertex sets |
| $w(i, j)$ | Data transmission latency between $V_i$ and $W_j$ in $G = (V \cup W; E)$ |
| $C$ | Set of all computation nodes |
| $C_{pM}$ | Set of pre-Map VMs |
| $t_1$ | Threshold for pre-Map VMs selection |
| $t_2$ | Threshold for Reduce VMs selection |
| $t_3$ | Maximum data transmission latency threshold |
| $t_4$ | Total data transmission latency threshold |
| $a_{ij}$ | Parameter with a value of 1 when $V_i$ is assigned to $W_j$; 0 otherwise |
| $G$ | $G = (V \cup W; E)$, where $V$ is the left vertex set, $W$ is the right vertex set, and $E$ is the edge set between the two vertex sets |
| $G'$ | $G' = (V' \cup W'; E')$ is a subgraph of $G$ with $V' \subset V$, $W' \subset W$, $E' \subset E$ |
| $G_M$ | Maximal perfect matching of G |
| $G'_M$ | $G'_M = (V'_M \cup W'_M; E'_M)$ is a matching of $G'$ |
| $w_{\min}$ | Minimum of $w(i, j)$ |
| $w_{\max}$ | Maximum of $w(i, j)$ |
| $w_{mid}$ | The minimal maximum data transmission latency |
| $w_{mid1,2}$ | The middle variable between $w_{min}$ and $w_{max}$ |
| $l_V[i]$ | Label for $V_i$ |
| $l_W[j]$ | Label for $W_j$ |
| $O$ | $O$=1 (if the objective is Object 1)<br>$O$=2 (if the objective is Object 2) |

We consider the data transmission latency under a distributed cloud environment. There are two resources in the datacenter: data nodes and computation nodes(VMs). Information is stored in the data nodes by cloud users. When the data is needed for computation, we should assign appropriate VMs for each data node. In our scenario, the distribution of data node is fixed. The only thing that we should consider is how to place the VMs for Map and Reduce layer, respectively. We assume that one VM can access data from one data node and only Map layer VMs

can accept data from the data nodes. VMs of the Reduce layer cannot receive data from the data nodes directly, but they can accept data from VMs in the Map layer. A VM cannot communicate with another VM in the same layer. Therefore, inter-VM latency only exists between the Map and Reduce VMs. In Section 3.1, we propose an algorithm which is used to minimize the total data transmission latency under data transmission latency threshold. Section 3.2 introduces another algorithm which minimizes maximum data transmission latency under total data transmission latency threshold. In Section 3.3, we describe an approach of VMs placement for Map and Reduce layer.

**3.1 Minimize the total data transmission latency**

The total data transmission latency has a deep influence on the total bandwidth cost. Cloud service provider and user both hope the bandwidth cost as low as possible. Minimizing the total data latency can effectively reduce the bandwidth cost. However, as a result of conflict between local optimization and global optimization, minimizing the total data transmission may increase the maximum data transmission latency of links. If this case occurs, job completion time will be delayed, and all the other tasks will be waiting for the slowest task. Hence, we must set a threshold for maximum data transmission latency of links to get a tradeoff. Links whose data transmission latency is above the threshold will be removed before the total data transmission latency optimization. As a result, we have two constraints: the maximum data transmission latency constraint and minimizing the total data transmission latency constraint.

Considering our focus is the data transmission latency among the nodes, we construct a bipartite graph $G = (V \cup W; E)$ to model the distribution and relationship of data nodes and computation nodes. In the bipartite graph $G = (V \cup W; E)$, $V$ is the left vertex set, it represents data nodes. $W$ is the right vertex set, it represents computation nodes, and $E$ is the edge set between the two vertex sets. $w(i, j)$ is the weight between $V_i$ and $W_j$, whose value equals to the data transmission latency between them. We assume the number of data nodes is $m$ ,and the number of computation nodes is $n$. Commonly, we have $m<n$. To simplify the problem, we increase the cardinality of data nodes to $n$, which makes the cardinalities of data nodes and computation nodes are same to each other. The dummy nodes have the weight (latency) of $\infty$ with other nodes.

In the mathematical discipline of graph theory, a matching or independent edge set in a graph $G = (V \cup W; E)$ is a set of edges without common vertices. It may also be an entire graph consisting of edges without common vertices. Specially, if each $V_i$ has a corresponding $W_j$ and vice verse, we call the matching as a perfect matching [26, 27]. Our aim is to get the perfect matching of $V$ and $W$ under the two constraints stated above.

Hungarian algorithm is a famous algorithm for bipartite graph matching. However, it can only get a maximum matching for the bipartite graph. Our aim is to find the matching which has the minimum sum of the data transmission latency. Thus, we propose an improved algorithm to match the bipartite graph $G$ and get the minimum total data transmission latency. For this, we construct a Linear Sum Assignment Problem (LSAP) [28, 29] to describe it. $w(i, j)$ is the latency (weight, edge) of $V_i$ and $W_j$, $a_{ij}$ only have two value. If $W_j$ is assigned to $V_i$, $a_{ij}$ has the value of 1. Otherwise, its value is 0. Formula (1) is minimizing the total data transmission latency. Equation (2) shows that each $W_j$ is assigned to a single $V_i$, and Equation (3) shows that each $V_i$ only accept one $W_j$ for matching. Equation (4) indicates that the value of $a_{ij}$ is either 1 or 0.

$$Minimize \ \sum_{i=1}^{n} \sum_{j=1}^{n} w(i, j) \cdot a_{ij} \tag{1}$$

subject to:

$$\sum_{i=1}^{n} a_{ij} = 1 \ for \ all \ j = 1, ..., n \tag{2}$$

$$\sum_{j=1}^{n} a_{ij} = 1 \ for \ all \ i = 1, ..., n \tag{3}$$

$$a_{ij} \in \{0,1\} \ for \ all \ i, j = 1, ..., n \tag{4}$$

Actually, Formula (1) is a minimal weight matching problem ("minimal weight" represents the minimal sum of the weights). Since each weight is positive number, the minimum sum of the weights equals to the maximum sum of the weights' negative values. Then, the problem is transformed to a maximal weight matching problem. For convenience, we mark each $V_i$ and $W_j$ with labels ($l_V[i]$ and $l_W[j]$), and any $w(i, j)$ satisfies $l_V[i] + l_W[j] \geqslant w(i, j)$ at any time in our proposed algorithm. By labeling $V_i$ and $W_j$ as $l_V[i]$ and $l_W[j]$, we can transform the maximal weight matching problem to maximal perfect matching problem. We use the proposed algorithm in this section to get the maximal perfect matching, which is the minimum total data transmission latency of bipartite graph. The algorithm is supported by the following theorem:

Theorem: *In the bipartite graph, if the sub-graph whose weights satisfy $l_V[i] + l_W[j] = w(i, j)$ (the sub-graph is called equal sub-graph) and have a perfect matching. Then, the perfect matching is the maximal weight matching of the bipartite graph.*

Proof: *For $G=(V \cup W; E)$,$G'=(V' \cup W'; E')$ is a sub-graph of G. $G'_M=(V'_M \cup W'_M; E'_M)$ is a matching of $G'$. If $G'_M$ is belong to a equal sub-graph, it satisfies the equation: $\sum w(i, j) = \sum(l_V[i] + l_W[j])$. However, if $G'_M$ have weight which is not belong to equal sub-graph, it satisfies the equation: $\sum w(i, j) < \sum(l_V[i] + l_W[j])$.Thus, the perfect matching of equal graph must be the maximal weight matching of the bipartite graph.*

We use the following steps to solve the transformed problem of Formula (1).

Step1: Firstly, to reduce the value of maximum data transmission latency, we reconstruct the bipartite graph $G = (V \cup W; E)$ to satisfy the threshold constraint. In the algorithm, we use $t$ to denote the maximum data transmission latency threshold. If $w(i, j) > t$, we pruning it out of the bipartite graph $G$. After this step, we get a new bipartite graph with all weights satisfying the maximum data transmission latency threshold. (Line 1-5)

Step2: We use the negative value of each weight, by which we transform the minimal weight matching to maximal weight matching. (Line 6)

Step3: We initialize the label value of $V_i$ and $W_j$. Since the problem has been changed to maximal weight matching problem, we initialize $l_V[i]$ with the maximum weight of $V_i$ and $W_j$ ($j$=1....$n$). We initialize $l_W[j]$ with 0. By doing this, we can make sure $l_V[i]+ l_W[j] \geqslant w(i, j)$. (Line 7-10)

Step4: We use Hungarian Algorithm to equal sub-graph of $G$ to find the maximum matching. We get a maximum matching $G_M$. (Line 11-12)

Step5: If $G_M$ is a perfect matching, the proposed algorithm is finished and $G_M$ is the minimum weight matching. (Line 13-14)

Step6: On the contrary, if $G_M$ isn't a perfect matching, we will revise the label $l_V[i]$ and $l_W[j]$ to enlarge the equal sub-graph. After the revising, it goes to Step4. We repeat Step4 - Step6 until we get the perfect matching. (Line 15-26)

**Algorithm 1:** Minimizing total latency with maximum latency threshold

**Input:** $G=(V \cup W; E)$ bipartite graph; maximum data transmission latency $t$; $w(i, j)$ is the weight between $V_i$ and $W_j$; $l_V[i]$ is the label of $V$, $l_W[j]$ is the label of $W$.
**Output:** bipartite matching $G_M$ with minimum total latency
**Note:** An alternating path is a path in which the edges belong alternatively to the matching and not to the matching.

1. **for** all $w(i, j)$ **do**
2. **if** $w(i, j)>t$
3. Pruning $w(i, j)$ out of $E$
4. **end if**
5. **end for**
6. $w(i, j)=-w(i, j)$ // we use negative value of each weight
7. **for** all $l_V[i]$, $l_W[j]$ **do**
8. $l_V[i]$=Max($w(i, j)$)
9. $l_W[j]=0$
10. **end for**
11. **for** $G=(l_V \cup l_W; E)$ **do**
12. use *Hungarian Algorithm* to match $G \rightarrow G_M$
13. **if** ( $G_M$ is a perfect matching where $l_V[i]+ l_W[j]= w(i, j)$)
14. **break**
15. **else**
16. **for** all ($l_V[i] \in$ alternating tree, $l_W[j] \notin$ alternating tree)
17. $d$=Min[$l_V[i]+ l_W[j]- w(i, j)$]
18. **end for**
19. **for** all $l_V[i]$ , $l_W[j] \in$ alternating tree
20. $l_V[i]= l_V[i]-d$
21. $l_W[j]= l_W[j]+d$
22. **end for**
23. **continue**
24. **end if**
25. **end for**
26. **Return** $G_M$

Through Algorithm1, we minimize the total data transmission latency with maximum data transmission latency threshold, which could maximize the profit of Cloud Service Provider at the same time.

### 3.2 Minimize maximum data transmission latency

Tasks are processed totally parallel in MapReduce. Thus, the total completion time of a job is determined by its slowest task. The slowest task always has maximum data transmission latency. Minimizing the maximum data transmission latency can tremendously decrease the completion time of job. However, minimizing the maximum data transmission can't ensure small total data transmission latency. They are conflict with each other. For the whole equality, we set a threshold to limit the total data transmission latency. Our aim is to minimize the maximum data transmission latency under the total data transmission latency threshold, which can help us to achieve a trade-off.

In this section, the bipartite graph is perfectly matched under the total latency threshold constraint. The problem can be described as Linear Bottleneck Assignment Problem (LBAP), which also known as a Bottleneck Maximum Cardinality Matching [30, 31]. Formula (5) is minimizing the maximum data transmission latency. Equation (6) shows that each $W_j$ is assigned to a single $V_i$, and Equation (7) shows that each $V_i$ only accept one $W_j$ for matching. Equation (8) indicates that the value of $a_{ij}$ is either 1 or 0. Equation (9) is the weight matrix.

$$Minimize \max_{i,j=1}^{i,j=n} w(i,j) \cdot a_{ij} \tag{5}$$

subject to

$$\sum_{i=1}^{n} a_{ij} = 1 \ for\ all\ \ j = 1,...,n \tag{6}$$

$$\sum_{j=1}^{n} a_{ij} = 1 \ for\ all\ i = 1,...,n \tag{7}$$

$$a_{ij} \in \{0,1\} \ for\ all\ i, j = 1,...,n \tag{8}$$

$$W = \{w(i, j)\}\ W\ is\ n \times n\ \ fixed\ coefficient\ matrix \tag{9}$$

To minimize maximum data transmission latency, we propose the following algorithm to solve this problem. The specific steps are shown as follows:

Step1: First, we sort the edges according to their weights $w(i, j)$, and order them as non-decreasing sequence. (Line 1)

Step2: We initialize the value of $w_{mid1}$, $w_{mid2}$, $w_{min}$, $w_{max}$, $w_{mid}$ and $E$ for binary search. (Line 2-7)

Step3: In this step, we use binary search to find the minimal maximum data transmission latency $w_{mid}$ of the perfect matching $G_M$. We first use Hungarian Algorithm to get the matching $G_M$. If $G_M$ is not a perfect matching, we revise the value of $w_{mid1}$ and $E$. Otherwise, we revise the value of $w_{mid2}$. (Line 8-19)

Step4: The $G_M$ which we get in step 3 is a perfect matching. Threshold $t$ is used to reduce the total data transmission latency of $G_M$. If its total transmission latency is above the threshold $t$, it will return an “error”. Or else,

$G_M$ is the bipartite matching with minimal maximum data transmission latency. (Line 20-24)

**Algorithm 2:**Minimizing maximum latency with total latency threshold

**Input**: $G=(V \cup W; E)$ bipartite graph; total data transmission latency *t*. $w(i, j)$ is the weight between $V_i$ and $W_j$;
**Output**: the minimal maximum latency $w_{mid}$

1. **Sort** all the $w(i, j)$ as a non-decreasing sequence
2. $w_{min}$=Min($w(i, j)$)
3. $w_{max}$=Max($w(i, j)$)
4. $w_{mid1}=w_{min}$
5. $w_{mid2}=w_{max}$
6. $w_{mid}$=0
7. $E=[w_{min}, (w_{mid1}+ w_{mid2})/2]$
8. **while** ($w_{mid1} < w_{mid2}$)
9. $w_{mid} = (w_{mid1} + w_{mid2})/2$
10. $E=[ w_{min}, w_{mid} ]$
11. use *Hungarian Algorithm* to match $G(V \cup W; E) \rightarrow G_M=(V_M \cup W_M; E_M)$
12. **if** ($G_M$ is not a perfect matching)
13. $w_{mid1}=w_{mid}$
14. **continue**
15. **else**
16. $w_{mid2}=w_{mid}$
17. **continue**
18. **end if**
19. **wend**
20. **if** (the total data transmission latency of $G_M > t$)
21. **Return** "error" //no bipartite matching satisfies our constraint.
22. **else**
23. **Return** $w_{mid}$
24. **end if**

Using Algorithm 2, we can minimize the maximum data transmission latency, which could enormously decrease the completion time of job.

### 3.3 VM placement for Map and Reduce

#### 3.3.1 VM classification

VM classification is extremely important to the overall optimization. Since our aim is to reduce data transmission latency between data nodes and VMs of Map layer, we should choose the VMs with short data transmission latency to the data node for Map layer firstly. As to the Reduce VMs, they are just required to be near the Map VMs since they only have communication with Map layer. The Reduce VMs placement is determined by the VMs placement of Map phase. By VM classification, we can get an accurate selection range of VMs which is called pre-Map. We get Map VMs from pre-Map through the algorithms stated above. The threshold $t_1$ which we use to get pre-Map is determined by cloud users. As shown in the following procedure, we classify VMs as the pre-Map layer $C_{pM}$ and the other part $C$-$C_{pM}$ based on their latency and threshold $t_1$. Since the relationship between the data nodes and the Map VMs is one-to-one, we add the computation node $C_j$ into $C_{pM}$ , when: $C_j \in C$, weight (or edge) $w(D_i, C_j) \leqslant t_1$. We keep doing this until no VM satisfies the threshold constraint. As a result, the VMs are classified to pre-Map and the remaining part.

#### 3.3.2 VM placement

The complete VM placement includes the Map phase and Reduce phase. In this section, we place VMs of Map and Reduce for two objectives. The first objective is minimizing the total data transmission latency in MapReduce, and the second is minimizing the maximum data transmission latency in Mapredcue. We use Algorithm 1 for the first objective and Algorithm 2 for the second objective. In our assumption, the Reduce phase starts after the Map phase. Thus, the Reduce VMs will be placed based on the VM placement of Map. The output of VM placement for Map is the input of VM placement for Reduce. In our scenario, the relationship between data nodes and Map VMs is one-to-one, and the relationship between Map and Reduce is many-to-many. The number of Reduce VMs is denoted as *k*, which is determined by the users. In the next paragraph, we start to place the VMs of the Reduce layer.

To determine the VM placement for the Reduce task, we propose the following method. Since the relationship between the Map and Reduce layers is many-to-many, we should make sure each VM of Reduce as near as possible to VMs of Map. We set a threshold $t_2$ to limit the data transmission latency between Map and Reduce. We choose VMs from the remaining set $C$-$C_M$ for the Reduce layer. If the weights(latency) between a VM of $C$-$C_M$ and VMs of Map layer are all below $t_2$, we add the VM into $C_R$ ( $C_R$ denotes the VMs set of Reduce). Finally, we adjust the value of $t_2$ until the number of VMs in $C_R$ reaches *k*.

#### 3.3.3 Procedure description

The whole steps of VMs placement for Map and Reduce is as follows:

Step1: We classify the VMs as pre-Map and the remaining part based on their weight (data transmission latency) with data nodes. (Line 1-8)

Step2: If the optimization objective is minimizing the total data transmission latency with maximum data transmission latency threshold $t_3$, we use Algorithm1 to place VMs for data nodes ($D$). We can get the VMs of Map layer ($C_M$) at the same time. Otherwise, we use Algorithm 2. (Line 9-13)

Step3: We place VMs for Reduce layer after the Map phase. We adjust $t_2$ until the number of $C_R$ equals to *k*. (Line 14-34)

**Procedure 1 :**VM placement for Map and Reduce

**Input:** $G=(D \cup C; E)$; threshold $t_1, t_2, t_3 t_4$;Object $O$

**Output:** the placed VMs of Map and Reduce**:** $C_M$ and $C_R$

**Note:** We use Algorithm1 for Objective 1, and Algorithm 2 for Objective 2

1. $C_{pM}$=0
2. **for** all $D_i \in D$ do
3. **for** all $C_j \in C$
4. **if** $w(i, j) \leqslant t_1$
5. $C_{pM}=C_{pM} \cup C_j$
6 **break**

7. **end if**
8. **end for**
9. **end for**
10. **if** ($O$==1)
11. use Algorithm 1 to [$G$=($D \cup C_{pM}$; $E$),$t_3$]→$G_M$=（$D \cup C_M$; $E_M$）
12. else
13. use Algorithm 2 to [$G$=($D \cup C_{pM}$; $E$),$t_4$]→$G_M$=（$D \cup C_M$; $E_M$）
14. **end if**
15. **for** decrease $t_2$
16. $C_R$=0
17. **for** all $C_j \in C$-$C_M$ **do**
18. $F$=1
19. **for** all $C_i \in C_M$ **do**
20. **if** $w(i, j)>t_2$
21. $F$=0
22. **break**
23. **end if**
24. **end for**
25. **if** ($F$==1)
26. $C_R=C_R \cup C_j$
27. **end if**
28. **end for**
29. **if** ($C_R > k$)
30. **continue**
31. **else**
32. **break**
33. **end if**
34. **end for**
35. **Return** $C_M$ and $C_R$

# 4 Experiments

In this section, we compare our proposed method with a conventional approach in terms of total data transmission latency and maximum data transmission latency. Moreover, we also study the parameters of our approach.

## 4.1 Experimental Setup

Our experiments are set up exactly the same way as the simulation setup of [4]. We set up a datacenter with 1024 racks and divide the racks into 64 blocks. Racks belonging to the same block can communicate with each other one switch. The 64 blocks are then divided into 16 groups. Racks belonging to the same group but not the same blocks can communicate with each other via three switches. Then, the 16 groups are divided into four parts. Racks that belong to the same part but not the same group can communicate with each other via five switches, whereas racks belonging to different parts can communicate with each other through seven switches. We assume the latency between a data node and VM as follows: To compare with the previous approach, we set the same experimental parameter with [4]. We get a random number in the range [0.75, 1.25], and multiply it by the number of switches between the data node and VM. In our simulation, the distribution of VMs is intensive when they are far from data nodes, and it is sparse when they are closer. We conducted various experiments for 16, 64, 256, and 1024 racks, and varied the number of data nodes from 10 to 80 multiples of 10. The number of VMs varied from 40 to 120 in steps of 5. Data nodes and VMs were placed into the racks at random. We compare our approach with a previous approach that places VMs with the triangle inequality constraint [4]. We denote the previous approach as VSTI (VM selection by triangle inequality). Our approach as VSMR (VM selection by Map and Reduce). As different number of data nodes has the similar trend, we show the experiment results of 40 data nodes in this paper.

## 4.2 VM distribution analysis

In this section, we compare the effect of our approach with the previous approach. Figure 2 and Figure 3 are diagrammatic sketches which simulate the distribution of data nodes and computation nodes. The x-axis and y-axis represent for an abstract range. The distance between two nodes is proportional with the transmission latency. In the figures, the squares represent the data nodes, and the small dots represent the computation nodes(VMs). The bigger rounds represent the VMs which are placed for the data nodes according to the placement algorithms. The two figures show the result that the total data transmission latency and maximum data transmission latency of Figure 3 are shorter than Figure 2. Since the VSTI takes more care on the inter-VM data transmission latency, it ignores the data transmission latency between data nodes and VMs.

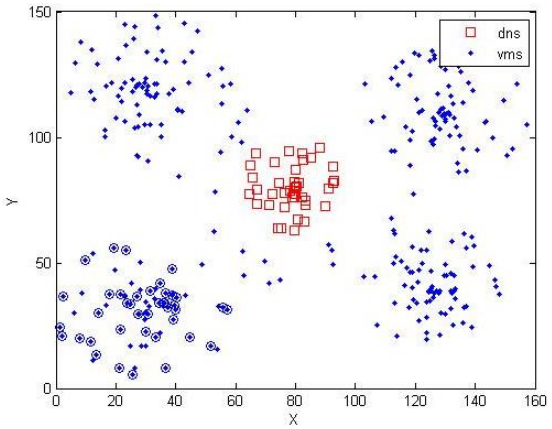


Figure.2 The VM placement result of VSTI. The distribution of VMs is intensive when they are far from data nodes and sparse when they are closer.

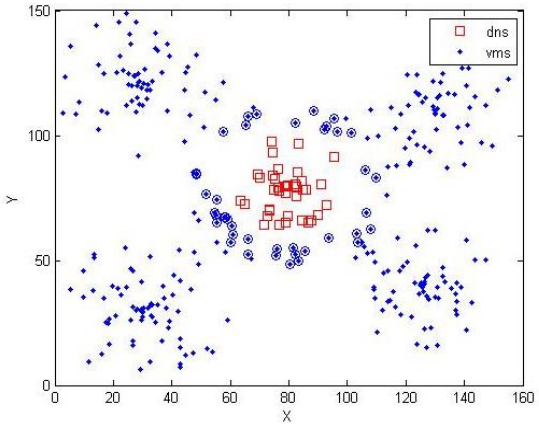


Figure.3 The VM placement result of VSMR. The distribution of VMs is intensive when they are far from data nodes and sparse when they are closer.

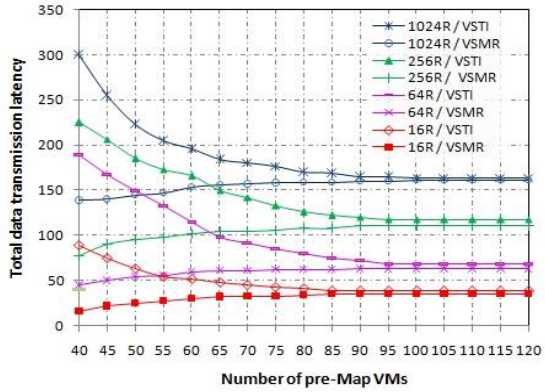


Figure.4 VSTI vs VSMR in minimizing total data transmission latency for different number of pre-Map VMs and 40 data nodes.

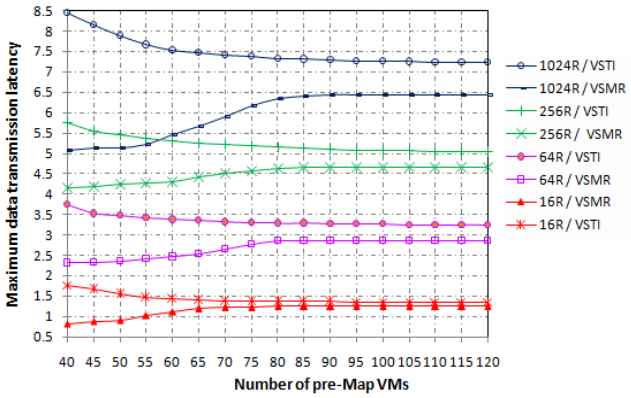


Figure.5 VSTI vs VSMR in minimizing maximum data transmission latency for different number of pre-Map VMs and 40 data nodes.

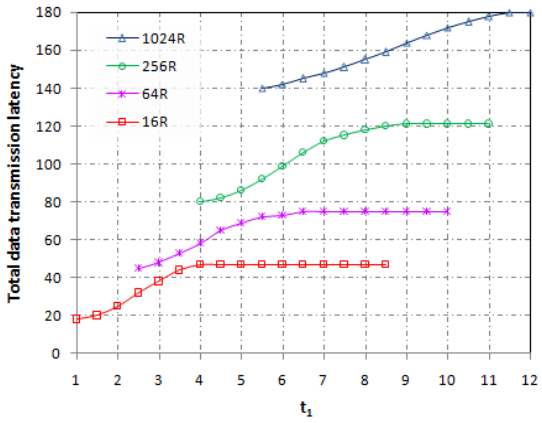


Figure.6 The total data transmission latency for different threshold $t_1$ using VSMR

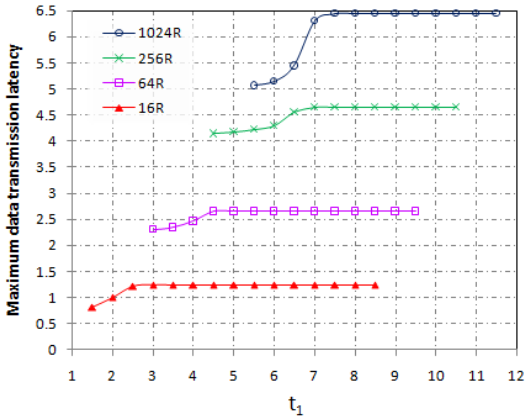


Figure.7 The maximum data transmission latency for different threshold $t_1$ using VSMR

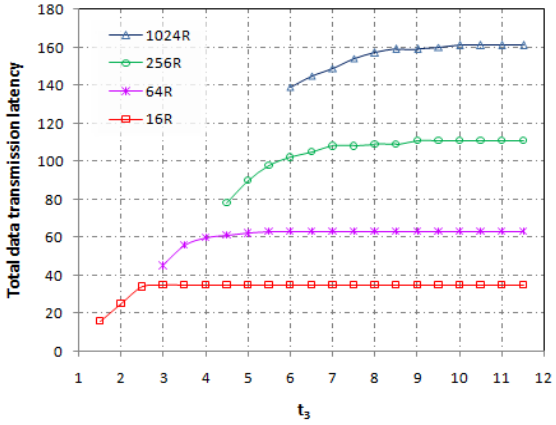


Figure.8 The total data transmission latency for different threshold $t_3$ using VSMR

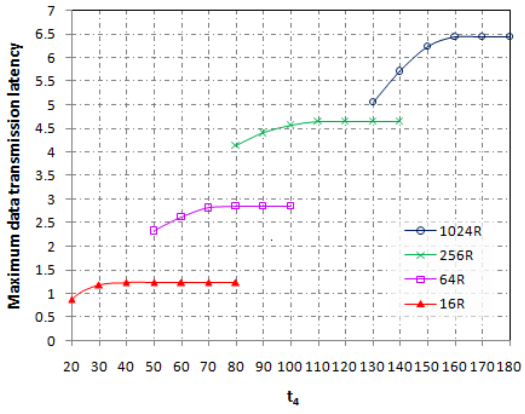


Figure.9 The trend of minimized maximum data transmission latency for different threshold $t_4$ using VSMR

Hence, the VSTI places the further VMs for data nodes in Figure 2. However, our approach focuses on the data transmission latency between data nodes and VMs, and places the near VMs for the data nodes. Then, the transmission latency of Figure 3 is shorter.

### 4.3 Total data transmission latency

Figure 4 shows that VSMR reduces the average total data transmission latency by 26.3% over VSTI when the number of pre-Map VMs ranges from 40 to 95. Since VSTI focuses on the inter-VM relationship, when the number of pre-Map VMs is small, it is impossible to choose all the VMs that are near to the data nodes. However, VSMR takes the relationship between the VM and data node as selection constraint for the pre-Map VMs. As a result, VSMR choose more near VMs than VSTI. When the number of pre-Map VMs ranges from 95 to 120, the cardinality of the pre-Map equals to the total number of VMs, and both approaches have the same total data

transmission latency. With the increase of the racks, since the range of VMs become larger, the total data transmission latency is higher.

### 4.4 Maximum data transmission latency

Figure 5 shows that VSMR reduces the maximum data transmission latency of VSTI by 41.2% when the number of pre-Map ranges from 40 to 95. Since VSTI is limited by the inter-VM constraint, many VMs that are sparsely distributed but near to data nodes are not chosen. Instead, VSTI chooses the pre-Map VMs that are sparsely distributed but far from the data nodes, which adversely affect the maximum data transmission latency. Since VSMR always choose the near VMs, VSMR is superior than VSTI. When the number of pre-Map VMs reaches 90, since the approaches already choose the nearest VMs for pre-Map, the maximum data transmission latency remains steady.

### 4.5 Study on parameters

In this section, we study the effect of parameters including $t_1$, $t_3$, and $t_4$ ($t_2$ doesn't affect the two objectives.) on total data transmission latency and maximum data transmission latency.

#### 4.5.1 Parameter $t_1$

In Figure 6, the total data transmission latency initially rises and then stabilizes. When $t_1$ is small, the VMs are not sufficient for data nodes. As the threshold $t_1$ increases, the bipartite graph can be perfectly matched. Since VSMR accurately obtains the closer VMs, the total data transmission latency remains steady, although $t_1$ continues to increase.

Figure 7 shows that the maximum data transmission latency increases continuously until $t_1$ is sufficiently large. As $t_1$ increases, the number of pre-Map VMs also increases. Then, VMs with high data transmission latency are added into the pre-Map set. This inevitably increases the maximum data transmission latency. After $t_1$ exceeds a certain value, the maximum data transmission latency remains unchanged.

#### 4.5.2 Parameter $t_3$

Figure 8 illustrates the relationship between the maximum data transmission latency threshold $t_3$ and the total data transmission latency. The total data transmission latency increases along with threshold $t_3$. As $t_3$ increases, the farther VMs are added into the pre-Map set. As a result, the total data transmission latency increases. When $t_3$ reaches a certain value, the total data transmission latency remains constant.

#### 4.5.3 Parameter $t_4$

Figure 9 shows that the maximum data transmission latency increases before stabilizing. This is because the farther VMs are added into the pre-Map VMs, which gives rise to higher maximum data transmission latency. In the case of 16 racks, when $t_4$ exceeds 30, the maximum data transmission latency remains constant. A similar case can be observed for 64, 256, and 1024 racks.

## 5 Conclusions

In this paper, we have proposed an approach to optimize data transmission latency through VM placement. Our approach first proposes two optimization algorithms to minimize the total data transmission latency and maximum data transmission latency. Then, we classify VMs to pre-Map and other parts based on their latency with the data nodes. We use the two algorithms to place VMs for Map phase. Finally, we place VMs for Reduce phase based on the inter-VM data transmission latency and VMs of Map phase. Compared to previous approaches, our approach has three main advantages. Firstly, the communication among VMs in our model is suitable for MapReduce. Secondly, the VMs with sparse distribution around the data nodes are efficiently utilized under MapReduce. Thirdly, our approach reduces the time complexity and data transmission latency enormously. The experimental results show that our approach reduces the average data transmission latency by 26.3% compared with other approaches.

However, the approach also can be improved further. The VM placement of Reduce phase can be more fine-grained through a new model. After this, the latency between Map phase and Reduce phase would be shorter.

**Acknowledgments**

The work presented in this study is supported by NSFC (61272521) and Research Fund for the Doctoral Program of Higher Education (20110005130001).

## References

[1] J. Dean, and S. Ghemawat, "MapReduce: simplified data processing on large clusters", *Communications of the ACM*, Vol.51, 2008, pp.107-113.

[2] J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for data intensive scientific analyses", *In: Proceedings of 4th IEEE International Conference on e-Science*, 2008, pp.277-284.

[3] T. Gunarathne, T.L. Wu, J. Qiu, and G. Fox, "MapReduce in the Clouds for Science", *In: Proceedings of 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010, pp.565-572.

[4] M. Alicherry, and T.V. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement", *In: Proceedings of 32nd IEEE International Conference on Computer Communications (INFOCOM)*, 2013, pp.647-655.

[5] J. Tordsson, R.S. Montero, R.M. Vozmediano, and I.M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers", *Future Generation Computer Systems*, Vol.28, 2012, pp.358-367.

[6] J.T. Piao, and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing", *In: Proceedings of 9th International Conference on Grid and Cooperative Computing (GCC)*, 2010, pp.87-92.

[7] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement", *In: Proceedings of IEEE International Conference on Services Computing (SCC)*, 2011, pp.72-79.

[8] M. Mao, and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows", *In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, 2011.

[9] S. Pandey, A. Barker, K.K. Gupta, and R. Buyya, "Minimizing execution costs when using globally distributed cloud services", *In: Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 2010, pp:222-229.

[10] Y. Feng, B. Li, and Bo Li, "Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward", *In: Proceedings of the 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2012, pp.43-50.

[11] A. Zhou, SG. Wang, QB. Sun, H. Zou, and FC. Yang, "Dynamic Virtual Resource Renting Approach for Maximizing the Profits of a Cloud Service Provider in a Dynamic Pricing Model", *In: Proceedings of 19th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2013, pp.118-125.

[12] Y.J. Hong, J. Xue, and M. Thottethodi, "Dynamic server provisioning to minimize cost in an IaaS cloud", *In: Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, ACM, 2011, pp.147-148.

[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system", *In: Proceedings of 26th International Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp.1-10.

[14] T. White, Hadoop: the definitive guide: the definitive guide, *O'Reilly Media*, Inc. 2009, pp.1-100.

[15] D. Borthakur, "The hadoop distributed file system: Architecture and design", *Hadoop Project Website*, Vol.11, 2007, pp.21-34.

[16] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments", *In: Proceedings of 8th USENIX Symposium on Operating Systems Design and Implementation(OSDI)*, Vol.8, 2008, pp.29-42.

[17] J. Lin, A. Bahety, S. Konda, and S. Mahindrakar, "Low-latency, high-throughput access to static global resources within the Hadoop framework", *University of Maryland, Tech. Rep*. 2009.

[18] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the Outliers in Map-Reduce Clusters using Mantri", *In: Proceedings of 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vol.10, 2010, pp.24-37.

[19] L. Simarro, J. Luis, R. M. Vozmediano, R.S. Montero, and I.M. Llorente, "Dynamic placement of virtual machines for cost optimization in multi-cloud environments", *In: Proceedings of IEEE International Conference on High Performance Computing and Simulation (HPCS)*, 2011, pp.1-7.

[20] N.M. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti, "VM placement strategies for cloud scenarios", *In: Proceedings of 5th IEEE International Conference on Cloud Computing (CLOUD)*, 2012, pp.852-859.

[21] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters", *In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles(SOSP)*, 2009, pp.261-276.

[22] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling", *In: Proceedings of the 5th European conference on Computer systems(EUROSYS)*, 2011, pp.265-278.

[23] A. Greenberg, J. Hamilton, D.A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks", *ACM SIGCOMM computer communication review*, Vol.39, 2008, pp.68-73.

[24] J.J. Kuo, H.H. Yang, and M.J. Tsai, "Optimal Approximation Algorithm of Virtual Machine Placement for Data Latency Minimization in Cloud Systems", *In: Proceedings of 33rd IEEE International Conference on Computer Communications (INFOCOM)*, 2014.

[25] R. Jonker, and T. Volgenant, "Improving the Hungarian assignment algorithm", *Operations Research Letters*, Vol.5, 1986, pp.171-175.

[26] D.W. Pentico, "Assignment problems: A golden anniversary survey", *European Journal of Operational Research*, Vol.176, 2007, pp.774-793.

[27] R. Jonker and A. Volgenant, "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems", *Computing, Springer-Verlag*, Vol.38, 1987, pp.325-340.

[28] R.E. Burkard, and E. Cela, "Linear assignment problems and extensions", *Handbook of Combinatorial Optimization*, Vol.3, 1999, pp.75-149.

[29] J.E. Hopcroft, and R.M. Karp, "An n^5/2 algorithm for maximum matchings in bipartite graphs", *In: Proceedings of 12nd IEEE Symposium on Foundations of Computer Science(FOCS)*, 1971, pp.122-125.

[30] H.N. Gabow, and R.E. Tarjan, "Algorithms for two bottleneck optimization problems", *Journal of Algorithms*, Vol. 9, 1988, pp.411-417.

[31] R.S. Garfinkel, "An Improved Algorithm for the Bottleneck Assignment Problem", *Institute for Operations Research and the Management Sciences (INFORMS)*, 1971, pp.1747-1751.