

Data Replica Location-Aware Joint Scheduling of Map and Reduce Tasks

Yuqi Fan ^{1†}, Bo Gao ²

1. School of Computer Science and Information Engineering, Anhui Province Key Laboratory of Industry Safety and Emergency Technology, Hefei University of Technology, Hefei, 230601, China

2. School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, 230601, China

Abstract

MapReduce is a popular data parallel processing framework in data centers. MapReduce splits a job into multiple map tasks and reduce tasks so that the tasks can be executed in parallel. Before running the map and reduce tasks, the task nodes communicate with the data nodes to fetch the data required by the execution of the tasks. The network traffic between the nodes accounts for a big part of the running time of the MapReduce job. Therefore, careful map and reduce tasks scheduling is critical for MapReduce performance. Most of the current task scheduling algorithms only perform the scheduling for either map tasks or reduce tasks without the joint consideration of the impact of both map and reduce tasks scheduling on the network traffic. In this paper, we deal with the joint scheduling of map and reduce tasks problem with the aim to reduce the network traffic. We also propose a data replica Location-Aware Joint Scheduling of map and reduce tasks algorithm (LAJS). The algorithm determines the scheduling locations of map and reduce tasks according to the node processing capabilities and the data replica locations of the input data for the map tasks. We finally conduct experiments through simulations. Experiment results show that the proposed algorithm LAJS can effectively reduce the data traffic during job processing and improve job makespan performance.

Keywords: MapReduce; Task scheduling; Data Location; Network Traffic

1 INTRODUCTION

Over the past decade, the rapid growth of Internet has produced increasingly more data. Highly scalable data-parallel frameworks are born to process massive data and tens of thousands of jobs. MapReduce^[1] is one of the popular data-parallel processing frameworks. Upon receiving a job submitted by a user, MapReduce splits the job into multiple maps and reduce tasks, deploys the tasks to some servers to compute in parallel, and merges the computation results which are returned to the user. The data required by the tasks are managed by Hadoop Distributed File System (HDFS)^[2].

The task nodes communicate with the data nodes to fetch the data before running the map and reduce tasks. Therefore, a large amount of network traffic is generated during the job processing. On one hand, each data required by the map tasks usually has three replicas in the data center, and the map tasks obtain the data from the nearest data nodes to reduce the network traffic according to data locality principle^[3]. On the other hand, a large amount of communication is required to transfer data from the map tasks to the reduce tasks during the shuffle phase, such that the reduce tasks can be executed^[4]. The traces from Facebook show that transferring data accounts for 33% of the job running time on average^[5]. Therefore, researchers try to reduce the network traffic by careful scheduling of map and reduce tasks.

The scheduling optimization for map tasks in MapReduce usually follows the principle of data locality. That is, the map tasks are scheduled to the locations with the required input data. The data locality enables the map tasks to read the input data locally without using the network to transfer a large number of data, thereby greatly reducing network traffic and improving MapReduce performance. The map task scheduling was modeled as a bipartite-graph matching problem, and a bipartite-graph oriented locality-aware scheduling algorithm was proposed to consider the divergence of node performance and distribution of data in MapReduce cluster^[6]. A map task scheduling algorithm was proposed

to improve the degree of data locality via an optimal map task selection scheduling strategy^[7]; when multiple tasks could meet the data locality principle in the same node, the algorithm would select the task which has the least number of replicas of input, individual load of disks attached to the node and the maximum expected time to wait for next local node. The map task scheduling problem was formulated as a job shop scheduling problem with the aim to reduce large-scale data movement and data transfer, and a prediction-based and locality-aware task scheduling algorithm (PLTS) was proposed^[8].

Each reduce task fetches the data from multiple map tasks, which involves significant amount of communication between the map tasks to the reduce tasks. Therefore, effective reduce task scheduling can improve MapReduce performance. A multi-level per node combiner and a dynamic ranking based MapReduce job scheduler were designed to keep intermediate records in memory without spilling into disks and place the reduce tasks on the virtual machines to minimize MapReduce job latency by exploiting dynamic performance of virtual machines^[9]. A data-local reduce task scheduling algorithm was proposed to schedule the reduce tasks to decrease the amount of network traffic by taking into consideration data locality^[10]. A center-of-gravity reduce scheduler (CoGRS) was proposed by considering the problems of data locality and partitioning skew, where the weighted total network distance of each reduce task was calculated and the node with the reduce task having the smallest WTND was the center-of-gravity node^[11].

Both map and reduce tasks scheduling has a significant impact on MapReduce performance. However, most of the current task scheduling algorithms only perform the scheduling for either map tasks or reduce tasks. In this paper, we tackle the problem of joint map and reduce task scheduling for the MapReduce jobs with the objective to minimize the network traffic by considering the data replica locations of the input data for map tasks.

The main contributions of this paper are as follows:

- (1) We investigate the task scheduling problem to reduce the network traffic with the joint consideration of the impact of both map task scheduling and reduce task deployment on MapReduce performance.
- (2) We formulate the task scheduling problem and propose a data replica Location-Aware Joint map and reduce tasks Scheduling algorithm (LAJS) to jointly deploy both map and reduce tasks for MapReduce jobs with the consideration of the data replica locations of the input data for map tasks.
- (3) We conduct simulations to evaluate the performance of the proposed algorithm. Simulation results demonstrate the proposed algorithm can effectively reduce the network traffic.

2 RELATED WORK

A MapReduce job consists of three phases: Map, Shuffle, and Reduce. For a given pair of nodes running map and reduce tasks, the network needs to send the data, e.g. the output of the reduce task, to the node executing the reduce tasks. Some research has been conducted on how to reduce the data transmission time of the shuffle phase to improve the MapReduce performance. The effect of flow scheduling strategies in information-agnostic datacenter networks is estimated and a priority queue based mathematical model was proposed to evaluate the performance of different strategies; the expression of flow completion time (FCT) was derived, and then several scheduling strategies for reducing the value of FCT were designed via the analysis on the expression^[12]. A heuristic algorithm called Fastest-Volume-Disposal-First was proposed to minimize coflow completion time (CCT); the algorithm performs flow-level traffic compression and scheduling system (Swallow) to reduce the amount of data transmitted over the network^[13]. A multiple-attributes-based coflow scheduling (MCS) mechanism was proposed to reduce the coflow completion time; at the start of a coflow, a shortest and narrowest coflow first algorithm was designed to assign the initial priority based on the coflow width; during the transmission of coflows, based on the sent bytes of coflows, a double-threshold scheme was proposed to adjust the priorities of different classes of coflows according to different thresholds^[14]. A responsive multipath TCP algorithm was proposed in SDN-based datacenters to improve the throughput of MapReduce with the SDN features; the algorithm employs a centralized controller for intelligent subflow route calculation and a monitor running on each server for actively adjusting the number of subflows^[15]. A dynamic network scheduler tags different types of traffic and utilizes Openflow to adjust transfers of flows dynamically^[16]. A network-aware shuffle scheduling algorithm (NAS) was proposed to minimize the makespan of MapReduce shuffle phase by leveraging the predictable

periodic network status^[17].

The scheduling optimization for map tasks in MapReduce usually follows the principle of data locality. That is, the map tasks are scheduled at the servers which have the required input data. The data locality enables the map tasks to read the input data locally without using the network to transfer a large number of data, thereby greatly reducing network traffic and improving MapReduce performance. The map task scheduling was modeled as a bipartite-graph matching problem, and a bipartite-graph oriented locality-aware scheduling algorithm (BOLAS) was proposed to improve the degree of data locality^[6]; the algorithm takes into account the divergence of node performance and distribution of data in MapReduce cluster, and employs the Kuhn-Munkres (KM) global minimal weighted matching algorithm to find the scheduling solution. An algorithm was proposed to improve the degree of data locality via an optimal map task selection scheduling strategy; when multiple tasks could meet the data locality principle in the same node, the algorithm would select the task which has the least number of replicas of input, individual load of disks attached to the node and the maximum expected time to wait for next local node^[7]. Based on the analysis of video decoding and encoding technologies and predicting the segment transcoding complexity, the map task scheduling problem was formulated as a job shop scheduling problem with the aim to reduce large-scale data movement and data transfer, and a prediction-based and locality-aware task scheduling algorithm (PLTS) was proposed^[8]. A map task scheduling algorithm and queueing architecture were presented; the algorithm was constituted by the Join the Shortest Queue policy together with the MaxWeight policy; an outer bound on the capacity region was identified and the number of backlogged tasks under the proposed algorithm was studied^[18].

Each reduce task fetches the data from multiple map tasks, and hence effective reduce task scheduling can improve MapReduce performance. A center-of-gravity reduce scheduler (CoGRS) was proposed by considering the problems of data locality and partitioning skew, where the weighted total network distance (WTND) of each reduce task was calculated and the node with the reduce task having the smallest WTND was the center-of-gravity node^[11]. A stochastic optimization framework to improve the data locality for Reduce Tasks was formulated, and a receding horizon control policy based on the optimal solution under restricted conditions was proposed^[19]. A reduce task scheduling algorithm was proposed to consider both data locality and network traffic, where network traffic awareness aims to distribute the traffic over the whole network and minimize the hotspots to reduce the effect of network congestion in data transfers^[20]. A data-local reduce task scheduling algorithm was proposed to schedule the reduce tasks to decrease the amount of network traffic by taking into consideration data locality^[10]. Each reduce task fetches the data from multiple map tasks, and effect reduce task scheduling can improve MapReduce performance. A multi-level per node combiner (MLPNC) and a dynamic ranking based MapReduce job scheduler (DRMJS) were designed^[9]; MLPNC used Memcache to keep intermediate records in memory without spilling into disks and used the rules proposed by PNC^[21] to merge the intermediate data so as to minimize the number of intermediate records; DRMJS placed the reduce tasks on the virtual machines to minimize MapReduce job latency by exploiting dynamic performance of virtual machines. A preemptive reduce task scheduling mechanism was proposed to correct the monopolizing behavior of long reduce tasks from large jobs, and a fair completion scheduler was proposed to dynamically balances the execution of different jobs for fair and fast completion^[22].

Both map and reduce tasks scheduling has a significant impact on MapReduce performance. However, most of the current task scheduling algorithms only perform the scheduling for either map tasks or reduce tasks. In this paper, we tackle the problem of joint map and reduce task scheduling for the MapReduce jobs with the objective to minimize the network traffic by considering the data replica locations of the input data for map tasks.

3 PROBLEM FORMULATION

The notations used in the paper are listed in Table 1.

TABLE 1 TABLE OF NOTATIONS

M	the set of map tasks
R	the set of reduce tasks
V	the set of nodes

m_i	i -th map task in set M
r_j	j -th reduce task in set R
v_k	k -th node in set V
v_i^m	the node running map task m_i
v_j^r	the node running reduce task r_j
w_k	the weight of node v_k
MD_k	the set of input data replicas of map tasks on node v_k
c_k^p	the processing capacity of node v_k
c_i^m	the processing capacity required to execute task m_i
c_j^r	the processing capacity required to execute task r_j
$p(v_k, v_{k'})$	the path length between nodes v_k and $v_{k'}$
$p(m_i, r_j)$	the path length between the nodes executing tasks m_i and r_j
$s(m_i, r_j)$	the size of data transferred between tasks m_i and r_j
$s(m_i)$	the size of input data of task m_i
$v_{i_1}, v_{i_2}, v_{i_3}$	the three nodes holding the input data replicas of tasks m_i
$x_{i,k}^m$	indicates whether map task m_i is deployed on node v_k (=1) or not (=0)
$x_{j,k}^r$	indicates whether reduce task r_j is deployed on node v_k (=1) or not (=0)
$y(v_k, v_{k'})$	indicates whether there is data transferred between nodes v_k and $v_{k'}$ (=1) or not (=0)

Assume that each MapReduce job is divided into a map task set M and a reduce task set R , where the processing capacities of map task m_i and reduce task r_j are c_i^m and c_j^r , respectively. The nodes running the map tasks and the reduce tasks are called *mapper* nodes and *reducer* nodes, respectively. Each data has three replicas with HDFS. Two of them are stored in the same rack, while the other replica is placed in a different rack. We assume the input data replicas of map task m_i are respectively distributed on nodes v_{i_1} , v_{i_2} and v_{i_3} . We need to decide the locations of each map and reduce task so as to decrease the network traffic required for fetching the input data for map tasks and reduce tasks, while satisfying the node processing capacity constraint.

The total network traffic incurred by fetching data from the data nodes for mapper node can be calculated as Eq. (1), and the total network traffic during the shuffle phase for transferring the data from the mapper nodes to the reducer nodes is computed via Eq. (2).

$$C^m = \sum_{i=1, i_n \in \{i_1, i_2, i_3\}, v_k \in V}^{i=|M|} x_{i,j}^m \cdot y_{v_k, v_{i_n}} \cdot s(m_i) \cdot p(v_{i_n}, v_k) \quad (1)$$

$$C^r = \sum_{i=1, j=1, v_k \in V, v_{k'} \in V}^{i=|M|, j=|R|} x_{i,j}^m \cdot x_{j,k'}^r \cdot y(v_k, v_{k'}) \cdot s(m_i, r_j) \cdot p(m_i, r_j) \quad (2)$$

In this paper, we aim to find an efficient strategy to jointly schedule map tasks and reduce tasks. That is, for the set of map tasks and the set of reduce tasks compose a MapReduce job, given a set of network nodes and the three data replica locations of each input data for each map task, our objective is to minimize the network traffic to fetch data so that the map and reduce tasks can be executed, under the constraint of node processing capabilities. In other words, our objective is to

Minimize

$$C = C^m + C^r \quad (3)$$

Subject to:

$$\sum_{k=1}^{k=|V|} x_{i,k}^m = 1, \forall m_i \in M \quad (4)$$

$$\sum_{k'=1}^{k'=|V|} x_{j,k'}^r = 1, \forall r_j \in R \quad (5)$$

$$\sum_{i=1}^{i=|M|} x_{i,k}^m \cdot c_i^m \leq c_k^p, \forall v_k \in V \quad (6)$$

$$\sum_{j=1}^{j=|R|} x_{j,k}^r \cdot c_j^r \leq c_k^v, \forall v_k \in V \quad (7)$$

$$x_{i,k}^m, x_{j,k}^r, y(v_k, v_{k'}) \in \{0,1\}, \forall v_k, v_{k'} \in V, \forall m_i \in M, \forall r_j \in R \quad (8)$$

Eq. (4) signifies that each map task is scheduled exactly once. Eq. (5) determines that each reduce task is deployed on one and only one node. Eq. (6) ensures the total processing capacity required to run the map tasks on node v_k does not exceed the processing capacity of node v_k . Eq. (7) ensures the total processing capacity required to execute the reduce tasks on node v_k does not exceed the processing capacity of node v_k . Eq. (8) dictates that $x_{i,k}^m, x_{j,k}^r$ and $y(v_k, v_{k'})$ are binary variables.

4 DATA REPLICA LOCATION-AWARE JOINT SCHEDULING OF MAP AND REDUCE TASKS

In this section, we propose a data replica Location-Aware Joint Scheduling of map and reduce tasks algorithm (LAJS) to jointly deploy the map and the reduce tasks for MapReduce jobs with the consideration of data replica locations of the input data of the map tasks. The map tasks will be scheduled at or close to the data nodes following the data locality principle so that the traffic between the mapper nodes and the data nodes is minimized. The scheduling of the reduce tasks is based on the relationship between the map tasks and the reduce tasks to minimize the network traffic across the nodes. To be specific, the reduce tasks are deployed at the nodes as close to the mapper nodes as possible to decrease the network traffic during the shuffle phase. Algorithm LAJS consists of three stages: (1) find the server racks for scheduling the map tasks, (2) schedule the map tasks, and (3) schedule the reduce tasks.

4.1 Find the Server Racks for Scheduling the Map Tasks

The objective of the algorithm is to minimize the network traffic, and thus we need to maximize the data locality of the map tasks. The algorithm also needs to schedule the reduce tasks close to the map tasks. Therefore, the algorithm schedules as many reduce tasks at the mapper nodes as possible. This stage determines the server racks for deploying the map and the reduce tasks according to the input data replicas locations of the map tasks.

For each rack, the algorithm calculates the number of input data which are required by the map tasks and stored on the rack. The algorithm then sorts the racks by the non-ascending order of the number of data. Note that each data has three data replicas. If two racks have the same number of input data, the algorithm sorts the two racks by descending order of the total number of input data replicas. The algorithm continues to find the racks for running the map tasks iteratively by finding a rack with largest number of input data in each iteration. The process continues until all the all the input data of the map tasks are found.

4.2 Schedule the Map Tasks

In this stage, algorithm LAJS shown in Algorithm 1 schedules the map tasks following the data locality principle so that the network traffic between the mapper nodes and data nodes is decreased.

Each data has three replicas. Two of them are stored in the same rack, while the other replica is placed in a different rack. There are multiple nodes in a rack, and each node may store the input data replicas of different map tasks. Algorithm LAJS finds the minimum number of nodes to schedule the map tasks, such that the data replicas on the found nodes compose the data set required by all the map tasks in set M under the constraint of node processing capacities. The set cover problem^[23], a well-known $\mathcal{NP}$ -complete problem, can be reduced to the map task scheduling problem without the consideration of the node processing capacity constraint, which is a special case of the map task scheduling problem. Therefore, the map task scheduling problem is also an $\mathcal{NP}$ -complete problem.

We propose a greedy algorithm to schedule the map tasks. We assign a weight for each node in the found racks at Stage 1. w_k , the weight of node v_k , is defined as the ratio between the total input data size on node v_k and the number of input data on node v_k . Algorithm chooses the node v_{k^*} with the biggest weight, i.e. $v_{k^*} = \arg \max_{v_k \in V} w_k$, to schedule the map tasks. The algorithm then sorts the data in the non-ascending order of the data sizes on node v_{k^*} . For each data on node v_{k^*} , the algorithm finds a node for the map task which takes the data as input data. If node v_{k^*} has enough processing capacity to run the task, the task is deployed on node v_{k^*} ; otherwise, if node $v_{k'}$ which has the second

replica of the data and is in the same rack as node v_{k^*} has enough processing capacity to execute the task, the task is scheduled on node $v_{k'}$. In case neither of the two nodes can accommodate the task, the task will be scheduled on node $v_{k''}$ which holds the third replica of the data, if node $v_{k''}$ is able of run the task; otherwise, a random node is chosen to execute the task and the node needs to fetch the data from one of the three nodes holding the data. The procedure continues until all the map tasks in set M are scheduled.

Algorithm 1 *Schedule the Map Tasks*

Input: The set of map tasks M ,

Input: The set of nodes V .

Output: The nodes used to schedule the map tasks.

1. **for** each node v_k **do**
 2. Calculate w_k ;
 3. **end for**
 4. **while** M is not empty **do**
 5. Choose node $v_{k^*} = \arg \max_{v_k \in V} w_k$;
 6. Sort the data in the non-ascending order of the data sizes on node v_{k^*} ;
 7. **for** each data on node v_{k^*} **do**
 8. **If** node v_{k^*} has enough processing capacity to run the task which takes the data as input data **then**
 9. Schedule the task on the node v_{k^*} ;
 10. **else if** node $v_{k'}$ which has the second replica of the data has enough processing capacity to execute the task **then**
 11. Schedule the task on node $v_{k'}$;
 12. **else if** node $v_{k''}$ which holds the third replica of the data has enough processing capacity to run the task **then**
 13. Schedule the task on node $v_{k''}$;
 14. **else**
 15. Choose a random node which has enough processing capacity to execute the task;
 16. **end if**
 17. Remove the task from M ;
 18. **end for**
 19. **end while**
 20. **return** the nodes used to run the map tasks.
-

4.3 Schedule the Reduce Tasks

Algorithm 2 illustrates the process of reduce task scheduling. We need to find an appropriate node for each reduce task so as to minimize the network traffic. Therefore, we want to schedule the reduce tasks on the nodes where the map tasks generate the data required by the reduce tasks. We create a bipartite graph based on the map tasks, the reduce tasks, and the association relationship between the tasks. Each map task can be abstracted as a node in the graph, and all the map tasks compose a part of the graph. Accordingly, all the reduce tasks constitute the other part of the graph. There is an edge between two nodes from the two parts, if there are some data to be transferred between the corresponding tasks. If map task m_i generates the input data of reduce task r_j , let $\beta(m_i, r_j) = \sum_{i'=1}^{i'=|M|} y(v_{i'}^m, v_i^m) \cdot p(v_{i'}^m, v_i^m) \cdot s(m_{i'}, r_j)$. The weight of the edge between two associated tasks m_i and r_j is defined as $\alpha - \beta(m_i, r_j)$, where $\alpha = \max_{m_i \in M, r_j \in R} \beta(m_i, r_j)$.

Algorithm 2 *Schedule the Reduce Tasks*

Input: The set of map tasks M ,

Input: The set of reduce tasks R ,

Input: The set of nodes V .

Output: The nodes used to schedule the reduce tasks.

1. Create a bipartite graph based on the map tasks, the reduce tasks, and the association relationship between the tasks;
-

```

2. for each edge between the nodes running  $m_i$  and  $r_j$  do
3.   Calculate  $\alpha - \beta(m_i, r_j)$  as the weight of the edge;
4. end for
5. for each node in the bipartite graph do
6.   Create the corresponding virtual nodes which number is the node degree minus 1;
7. end for
8. Employ KM global minimal weighted matching algorithm to obtain the scheduling solution;
9. Sort the reduce tasks in the non-increasing order of the required input data sizes;
10. while  $R$  is not empty do
11.   for each reduce task  $r_j \in R$  do
12.     if the node found by the KM algorithm has enough processing capacity to process  $r_j$  then
13.       Schedule  $r_j$  on the node;
14.     else if another node in the same rack has enough processing capacity to execute  $r_j$  then
15.       Schedule  $r_j$  on the node;
16.     else
17.       Choose a random node which has enough processing capacity in the racks found in Stage 1 to
         execute  $r_j$ ;
18.     end if
19.     Remove  $r_j$  from  $R$ ;
20.   end for
21. end while
22. return the nodes used to schedule the reduce tasks.

```

We then create virtual nodes in the bipartite graph so that the two parts of the graph have the same number of nodes. For each node (real node), the number of corresponding virtual nodes is the node degree minus 1. There is an edge between two nodes if there is an edge between the corresponding real nodes. The edge weight is ∞ if one or both of the two end nodes of the edge are virtual nodes.

We employ the Kuhn-Munkres (KM) global minimal weighted matching algorithm to obtain the scheduling solution without the consideration of the node processing capacity. We then sort the reduce tasks in the non-increasing order of the required input data sizes. For each reduce task, we check whether the node found by the KM algorithm has enough processing capacity to process the reduce task. If no, we check whether another node in the same rack is able to run the reduce task. If no, we find another node with enough processing capacity in the racks found in stage 1 to execute the reduce task.

4.4 Time Complexity

Algorithm LAJS consists of three stages: (1) find the server racks for scheduling the map tasks, (2) schedule the map tasks, and (3) schedule the reduce tasks. Assume that N_R and N_S are the number of racks in the network and the number of servers in each rack, respectively. Denote $|M|$ and $|R|$ are the number of map tasks and the number of reduce tasks, respectively. When determining the racks for deploying the map tasks and the reduce tasks, sorting the racks in the network takes $O(N_R \log(N_R))$ time. During the process of scheduling the map tasks, calculating the node weights takes $O(N_S|M|)$ time, sorting the nodes takes $O(N_S \log(N_S))$ time, and finding the node for each map task takes $O(|M|)$ time. Therefore, it takes $O(N_S|M|)$ time to schedule the map tasks. When scheduling the reduce tasks, constructing the bipartite graph takes $O(|M||R|)$ time, adding virtual nodes in the bipartite graph takes $O(|M| + |R|)$ time, calculating the edge weights takes $O(M|R|)$ time, the KM algorithm takes $O(M^2|R|^2)$ times, and finding the node for each reduce task takes $O(|R|)$ time. Therefore, it takes $O(M^2|R|^2)$ time to schedule the reduce tasks. In summary, the time complexity of algorithm LAJS is $O(M^2|R|^2)$.

5 CONCLUSIONS

In this section, we evaluate the performance of the proposed algorithm LAJS. We also investigate the impact of important parameters on the performance of the proposed algorithm.

5.1 Simulation setup

We evaluate the proposed algorithm LAJS in terms of the total network traffic, NTRP and MapReduce job makespan against the state-of-the-arts: BOLAS^[6] and RTS^[10]. The network traffic incurred by transferring a data is the product of the data size and the data transmission path length. NTRP (Network Traffic Reduction Percentage) is the percentage of the total network traffic reduction of algorithm LAJS over the two benchmark algorithms BOLAS and RTS. Algorithm BOLAS models the map task scheduling problem as a bipartite-graph matching problem^[24] by considering the node performance divergence and data distribution in MapReduce clusters, and employs the KM global minimal weighted matching algorithm to obtain the optimal scheduling solution to achieve high data locality degree. Algorithm RTS schedules the reduce tasks on the nodes that will result in the least amount of network traffic by considering data locality. Note that algorithm BOLAS only schedules the map tasks, while algorithm RTS only deploys the reduce tasks. Neither of the two benchmark algorithms considers the joint scheduling of the map tasks and the reduce tasks.

The network topology used in the simulation is a fat tree which is commonly used in data centers^[25]. With the fat tree, the path length between a node pair is as follows: if the two nodes are in the same rack, the path length is 2; if the two nodes are in different racks but in the same pod, the path length is 4; if the two nodes are in different pods, the path length is 6. The node processing capabilities are randomly generated from 100-300, and the processing capacity required by each task is randomly generated from 1-10. Since algorithm BOLAS only schedules the map tasks, we assume algorithm BOLAS uses the default MapReduce scheduling solution during scheduling the reduce tasks. Similarly, we assume algorithm RTS schedules the map tasks with the default MapReduce scheduling solution. MapReduce schedules a reduce task on a random idle server, and deploys a map task on an idle server which is closest to the node which holds the input data of the task among all the idle servers^[26]. The simulation studies the impact of three different parameters on algorithm LAJS, including the number of map tasks, the number of reduce tasks, and the number of nodes. We conduct the simulations for 120 runs, and the average value of the running results is taken as the final result.

5.2 Performance Evaluation of the Proposed Algorithm

1) Impact of various number of map tasks

We evaluate the performance of different algorithms of LAJS, BOLAS and RTS by varying the number of map tasks, assuming there are 12 nodes and 6 reduce tasks. Similar to algorithms BOLAS and RTS, the number of map tasks increases from 10 to 160.

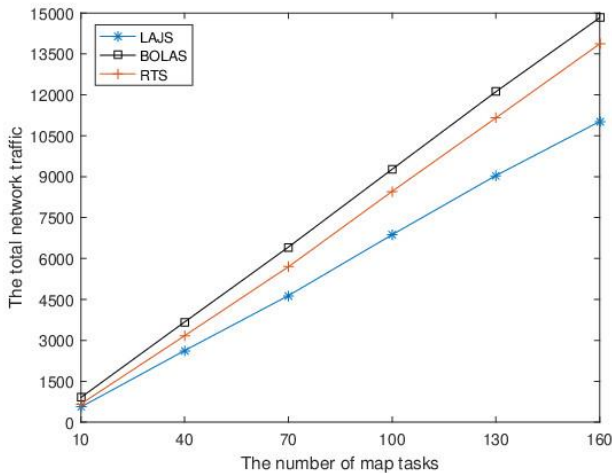


FIG. 1 THE TOTAL NETWORK TRAFFIC UNDER DIFFERENT NUMBER OF MAP TASKS

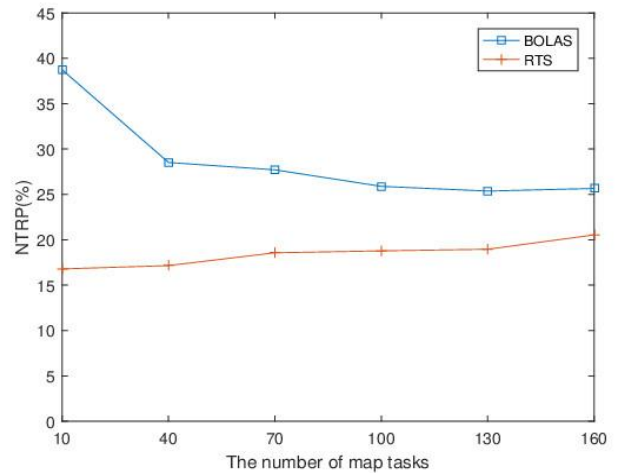


FIG. 2 NTRP UNDER DIFFERENT NUMBER OF MAP TASKS

Figs. 1 and 2 show the network traffic and NTRP for different algorithms with different number of map tasks, respectively. The network traffic of all the three algorithms increases when the number of map tasks increases, because the total number of data fetched by the map and reduce tasks increases with the increasing number of map tasks. Algorithm LAJS achieves the best performance among the three algorithms, since algorithm LAJS considers the scheduling of both the map and the reduce tasks to decrease the network traffic. Algorithm LAJS outperforms

algorithm RTS from about 17% to 20% with the number of map tasks increasing from 10 to 160, since algorithm RTS reduces the network traffic from the only perspective of the reduce task scheduling. The NTRP of algorithm BOLAS decreases as the number of map tasks increases, because more map tasks can get the data from the locations close to the data nodes. The NTRP of algorithm RTS slightly increases with the increasing number of mapping tasks, since more data transmission is required between the map and the reduce tasks.

Fig. 3 demonstrates the job makespan under different number of map tasks. The makespan of all the three algorithms increases when the number of map tasks increases, because more map tasks are executed on each node and more data are transmitted during the job processing with the increasing number of map tasks. It can be observed that algorithm LAJS takes the shortest time to run the jobs, which shows that algorithm LAJS can improve the performance of MapReduce by joint map and reduce task scheduling.

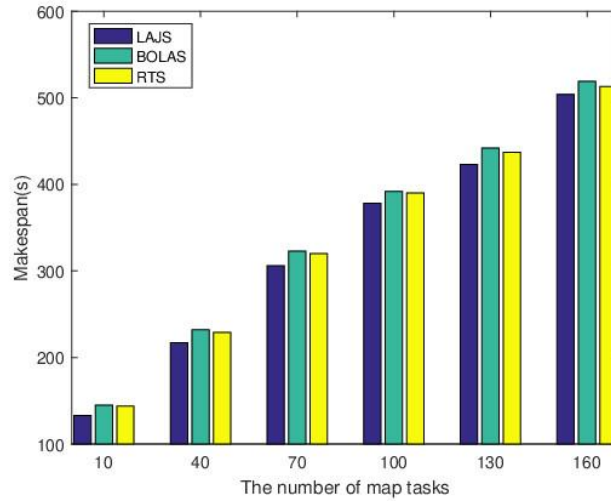


FIG. 3 MAKESPAN UNDER DIFFERENT NUMBER OF MAP TASKS

2) Impact of various number of reduce tasks

We study the impact of various number of reduce tasks on the performance of different algorithms. Similar to algorithms BOLAS and RTS, the number of reduce tasks increases from 2 to 12, and there are 12 nodes and 100 map tasks.

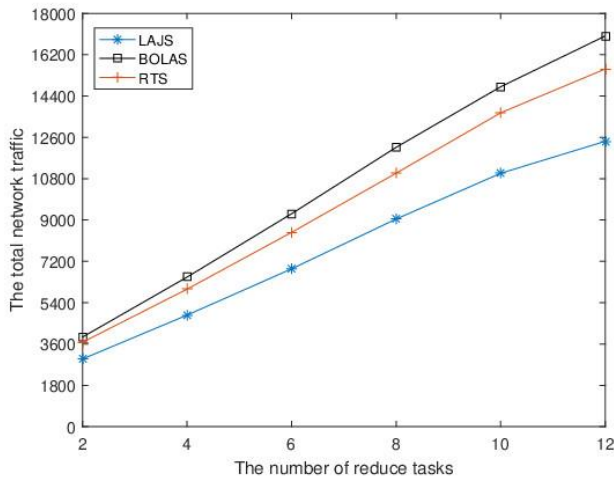


FIG. 4 THE NETWORK TRAFFIC UNDER DIFFERENT NUMBER OF REDUCE TASKS

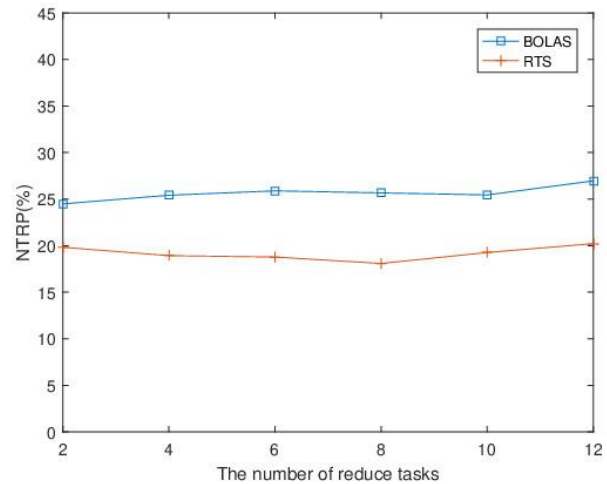


FIG. 5 THE NTRP UNDER DIFFERENT NUMBER OF REDUCE TASKS

Figs. 4 and 5 illustrate the total network traffic and NTRP for different algorithms by varying the number of reduce tasks. As the number of reduce tasks increases, the network traffic of all the three algorithms increases. More reduce tasks lead to more map and reduce task node pairs, which increases the amount of network traffic during the shuffle

phase. Algorithm LAJS performs the best among the three algorithms, and the performance improvement of algorithm LAJS over the two benchmark algorithms keeps pretty stable. To be specific, algorithm LAJS outperforms algorithm BOLAS from about 24% to 27% and performs better than algorithm RTS about 19.5% with the number of reduce tasks increasing from 2 to 12. Algorithm LAJS jointly schedules the map and the reduce tasks, and carefully deploys the reduce tasks to optimize the network traffic from the mapper nodes to the reducer nodes. Algorithm BOLAS decreases the network traffic by only scheduling map tasks, while algorithm RTS only focuses on the reduce task scheduling.

Fig. 6 shows the job makespan under different number of reduce tasks. The makespan of all the three algorithms decreases with the increase of the number of reduce tasks. The data sizes between the reduce tasks and the map tasks are potentially smaller, and hence the transmission time during the shuffle phase is shorter with the increasing number of reduce tasks. LAJS algorithm is significantly better than the other two algorithms, which shows that the optimization of network traffic for task execution can improve the MapReduce job performance.

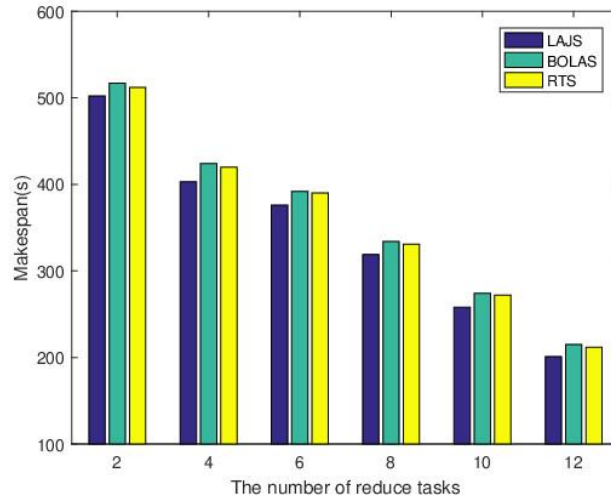


FIG. 6 MAKESPAN UNDER DIFFERENT NUMBER OF REDUCE TASKS

3) Impact of various number of nodes

We study the impact of various number of nodes on the performance of different algorithms. Assume that there are 100 map tasks and 6 reduce tasks. The number of nodes in the network increases from 6 to 16.

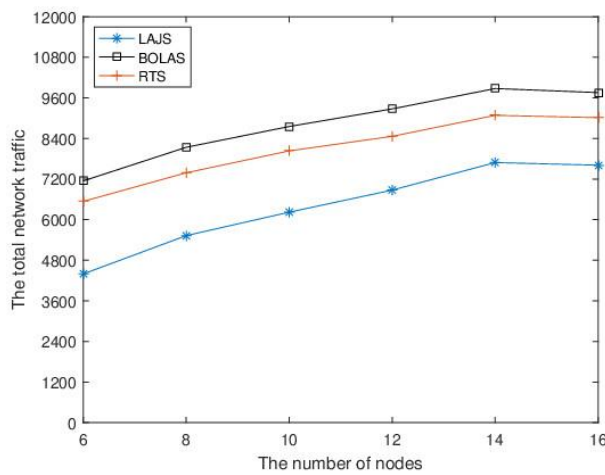


FIG. 7 THE NETWORK TRAFFIC UNDER DIFFERENT NUMBER OF NODES

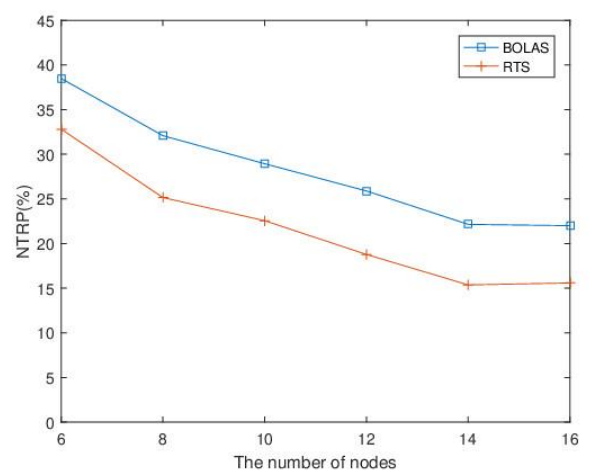


FIG. 8 THE NTRP UNDER DIFFERENT NUMBER OF NODES

Figs. 7 and 8 demonstrate that the total network traffic and NTRP for different algorithms by varying the number of nodes. The network traffic of all the three algorithms increases when the number of nodes increases. With a large number of nodes, the locations of the input data are also widely distributed, the number of racks and servers involved

in the tasks deployment increases, and hence more network traffic is required to get the data for the map and reduce tasks. The NTRP of both BLOAS and RTS decreases as the number of nodes increases. With a large number of nodes in the network, it is hard for all the algorithms to find the nodes to run the tasks so that both the map tasks and the reduce tasks can be located at or close to the required data.

Fig. 9 shows the job makespan under different number of nodes. The makespan of all the three algorithms decreases as the number of nodes increases, because the number of tasks executed on each node decreases with the increasing number of nodes. Algorithm LAJS performs the best among the three different algorithms, since algorithm LAJS considers all the performance impact of the map and reduce tasks and the network traffic during the job processing.

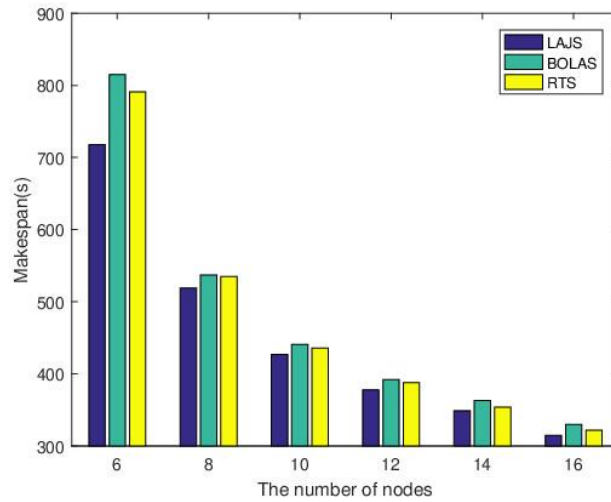


FIG. 3 MAKESPAN UNDER DIFFERENT NUMBER OF NODES

It can be observed from Figs. 1-9 that algorithm RTS performs better than algorithm BALOS, which shows that the reduce tasks scheduling plays an important role in the task deployment process. Normally, the number of reduce tasks is smaller than the number of map tasks and the number of input data locations. A large amount of communication is required to transmit the data from the mapper nodes to the reducer nodes during the shuffle phase. Therefore, carefully scheduling the reduce tasks can improve the MapReduce performance.

6 CONCLUSIONS

MapReduce splits a job into multiple map tasks and reduce tasks so that the tasks can be executed in parallel. Before running the map tasks and the reduce tasks, the task nodes communicate with the data nodes to fetch the data required by the execution of the tasks. Therefore, careful scheduling of the map tasks and the reduce tasks can greatly improve the performance of the MapReduce jobs. Most of the current task scheduling algorithms only performed the scheduling for either map tasks or reduce tasks without the joint consideration of the impact of both map and reduce tasks scheduling on the network traffic. In this paper, we formulated the joint scheduling of map and reduce tasks problem with the aim to minimize the network traffic. We proposed a data replica Location-Aware Joint map and reduce Scheduling algorithm (LAJS) to reduce the network traffic. The algorithm determined the scheduling locations of map tasks and reduce tasks according to the node processing capabilities and the data replica locations of the input data for the map tasks. We finally conducted experiments through simulations, and the experimental results demonstrated the proposed algorithm could effectively reduce the network traffic and job makespan.

REFERENCES

- [1] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.
- [2] Apache Software Foundation, "HDFS Users Guide," Hadoop Documentation. Accessed Dec 13, 2024. <http://Hadoop.apache.org/docs/stable/Hadoop-project-dist/Hadoop-HDFS/HDFSUserGuide.html>.

- [3] Song, Jie, Sun Zongzhe, Mao Keming, Bao Yubin, and Yu Ge. "Research advance on MapReduce based big data processing platforms and algorithms." *Journal of Software* 28.3 (2017): 514-543.
- [4] Babu, LD Dhinesh, and P. Venkata Krishna. "An execution environment oriented approach for scheduling dependent tasks of cloud computing workflows." *International Journal of Cloud Computing* 3.2 (2014): 209-224.
- [5] Guo, Yanfei, Rao Jia, Cheng Dazhao, and Zhou Xiaobo. "ishuffle: Improving hadoop performance with shuffle-on-write." *IEEE transactions on parallel and distributed systems* 28.6 (2016): 1649-1662.
- [6] Xue, Ruini, Gao Shengli, Ao Lixiang, and Guan Zhongyang. "BOLAS: bipartite-graph oriented locality-aware scheduling for MapReduce tasks." *2015 14th International Symposium on Parallel and Distributed Computing*. IEEE, 2015.
- [7] Lydia, E. Laxmi, Ch Sudhakar, and T. Madhuri. "An improved optimal task selection strategy for hadoop scheduling." *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. IEEE, 2017.
- [8] Zhao, Hui, Zheng Qinghua, Zhang Weizhan, and Wang Jing. "Prediction-based and locality-aware task scheduling for parallelizing video transcoding over heterogeneous mapreduce cluster." *IEEE Transactions on Circuits and Systems for Video Technology* 28.4 (2016): 1009-1020.
- [9] Jeyaraj, Rathinaraja, and V. S. Ananthanarayana. "Dynamic performance aware reduce task scheduling in MapReduce on virtualized environment." *2018 IEEE 16th International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE, 2018.
- [10] Geetha, J., N. UdayBhaskar, and P. ChennaReddy. "Data-local reduce task scheduling." *Procedia Computer Science* 85 (2016): 598-605.
- [11] Hammoud, Mohammad, M. Suhail Rehman, and Majd F. Sakr. "Center-of-gravity reduce task scheduling to lower mapreduce network traffic." *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012.
- [12] Hu, Chao, Liu Bo, Xing Changyou, Yue Zhenjun, Song Lihua, and Chen Ming. "Queueing model based analysis on flow scheduling in information-agnostic datacenter networks." *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016.
- [13] Zhou, Qihua, Li Peng, Wang Kun, Zeng Deze, Guo Song, and Guo Minyi. "Swallow: Joint online scheduling and coflow compression in datacenter networks." *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018.
- [14] Wang, Shuo, Zhang Jiao, Huang Tao, Pan Tian, Liu Jiang, and Liu Yunjie. "Multi-attributes-based coflow scheduling without prior knowledge." *IEEE/ACM Transactions on Networking* 26.4 (2018): 1962-1975.
- [15] Duan, Jingpu, Zhi Wang, and Chuan Wu. "Responsive multipath TCP in SDN-based datacenters." *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015.
- [16] Li, Zhao, Shen Yao, Yao Bin, and Guo Minyi. "OFScheduler: a dynamic network optimizer for MapReduce in heterogeneous cluster." *International Journal of Parallel Programming* 43 (2015): 472-488.
- [17] Fan, Yuqi, Liu Wenlong, Guo Dan, Wu Weili, and Du Dingzhu. "Shuffle scheduling for MapReduce jobs based on periodic network status." *IEEE/ACM Transactions on Networking* 28.4 (2020): 1832-1844.
- [18] Wang, Weina, Zhu Kai, Ying Lei, Tan Jian, and Zhang Li. "Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality." *IEEE/ACM Transactions On Networking* 24.1 (2014): 190-203.
- [19] Tan, Jian, Meng Shicong, Meng Xiaoqiao, and Zhang Li. "Improving reducetask data locality for sequential mapreduce jobs." *2013 Proceedings IEEE INFOCOM*. IEEE, 2013.
- [20] Arslan, Engin, Mrigank Shekhar, and Tefvik Kosar. "Locality and network-aware reduce task scheduling for data-intensive applications." *2014 5th International Workshop on Data-Intensive Computing in the Clouds*. IEEE, 2014.
- [21] Lee, Woo-Hyun, Hee-Gook Jun, and Hyoung-Joo Kim. "Hadoop Mapreduce performance enhancement using in-node combiners." *arXiv preprint arXiv:1511.04861* (2015).
- [22] Wang, Yandong, Tan Jian, Yu Weikuan, Zhang Li, Meng Xiaoqiao, and Li, Xiaobing. "Preemptive ReduceTask Scheduling for Fair and Fast Job Completion." *10th International Conference on Autonomic Computing (ICAC)* 13. 2013.
- [23] Levin, Asaf. "Approximating the unweighted k-set cover problem: greedy meets local search." *SIAM Journal on Discrete Mathematics* 23.1 (2009): 251-264.
- [24] Tanimoto, Steven L., Alon Itai, and Michael Rodeh. "Some matching problems for bipartite graphs." *Journal of the ACM (JACM)* 25.4 (1978): 517-525.
- [25] Cheng, Li-Wei, and Shie-Yuan Wang. "Application-aware SDN routing for big data networking." *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015.
- [26] Apache Software Foundation, "MapReduce Tutorial," Hadoop Documentation. Accessed Dec 13, 2024. <http://hadoop.apache.org/docs/r3.2.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Mapper>