Shuffle Scheduling for MapReduce Jobs Based on Periodic Network Status

Yuqi Fan, Wenlong Liu, Dan Guo, Weili Wu, and Dingzhu Du

***Abstract*— MapReduce jobs need to shuffle a large amount of data over the network between mapper and reducer nodes. The shuffle time accounts for a big part of the total running time of the MapReduce jobs. Therefore, optimizing the makespan of shuffle phase can greatly improve the performance of MapReduce jobs. A large fraction of production jobs in data centers are recurring with predictable characteristics, and the recurring jobs split the network into periodic busy and idle time slots, which allows us to better schedule the shuffle data in order to reduce the makespan of shuffle phase with the future predictable network status available. In this paper, we formulate the shuffle scheduling problem with the aim to minimize the makespan of MapReduce shuffle phase by leveraging the predictable periodic network status. We then propose a simple yet effective network-aware shuffle scheduling algorithm (NAS) to reduce the number of idle time slots required to transfer the shuffle data so as to reduce the shuffle makespan. We also prove that the proposed algorithm NAS is a $\frac{3}{2}$-approximation algorithm to the shuffle scheduling problem when all the future idle time slots have the same duration. We finally conduct experiments through simulations. Experimental results demonstrate the proposed algorithm can effectively reduce the makespan of MapReduce shuffle phase and increase network utilization.**

***Index Terms*— MapReduce, network-aware, shuffle scheduling, makespan.**

I. 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 a popular data-parallel processing framework proposed by Google in 2004, and has been deployed by many firms, such as Google, Facebook, Yahoo!, etc. The basic rationale of MapReduce is to split jobs submitted by users into multiple map and reduce tasks, assign the tasks to some servers to compute, and merge the computation results which are returned to the users. A MapReduce job consists of three phases: Map, Shuffle, and Reduce. The nodes running the map tasks and the reduce tasks are called *mapper* nodes and *reducer* nodes, respectively. During the shuffle phase, significant amount of communication is required from the mapper nodes to the reducer nodes so that the reduce tasks can be executed [2]. The traces from Facebook show that transferring data accounts for 33% of the job running time on average [3].

Researchers try to reduce the time spent in the shuffle phase to improve the performance of MapReduce by network resources scheduling [4], [5], task scheduling [6], [7], and data flow scheduling [8], [9]. Network resource scheduling optimizes MapReduce by careful network bandwidth allocation, network load balancing, etc. Task scheduling is engaged in map tasks scheduling and reduce tasks scheduling. Map tasks scheduling usually achieves data locality to reduce the network traffic required for fetching data from the data storage nodes before running map tasks. Similarly, the scheduling of reduce tasks generally selects appropriate nodes to reduce the network transmission from the mapper nodes to the reducer nodes. Data flow scheduling transfers the data between the nodes based on data priority, data categories, real-time demand, etc.

After the mapper and reducer nodes are allocated, data transmission is required between the nodes so that the reducer nodes can obtain the data to run the reduce tasks. Network status has a direct and significant impact on the performance of data flow scheduling in the shuffle phase. A good understanding of future network status enables us to plan the data transmission in advance, such that we can effectively improve the performance of MapReduce shuffle scheduling by making good use of the network resources as demonstrated in Example. 1.

Example 1: As shown in Fig. 1, four shuffle data items are to be transmitted, and the data items are generated one by one in turn. The sizes of the data items are 8, 5, 11, and 2, respectively. Assume there exist 4 *future idle time slots with sizes of 12, 10, and 6, and 12, respectively. Without knowing the future network status, the four data items will be scheduled in turn. That is, the first and the second data items will be scheduled in the first two time slots, respectively; the third data item will be transmitted in the fourth time slot, since the first three time slots have no enough idle resource for the third data item; the fourth data item will be scheduled in the first time slot. The makespan of shuffle scheduling will be 71. If we consider the future network status, we can schedule*

Manuscript received March 31, 2019; revised September 18, 2019, January 3, 2020, February 2, 2020, and April 17, 2020; accepted May 8, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor H. Shen. This work was supported in part by the National Natural Science Foundation of China under Grant 61701162, in part by the U.S. National Science Foundation under Grant 1747818 and Grant 1907472, and in part by the Anhui Provincial Natural Science Foundation under Grant 1608085MF142. *(Corresponding author: Yuqi Fan.)*

Yuqi Fan, Wenlong Liu, and Dan Guo are with the Key Laboratory of Knowledge Engineering with Big Data, Ministry of Education, Hefei University of Technology, Hefei 230601, China, and also with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China (e-mail: yuqi.fan@hfut.edu.cn; lamwolog@gmail.com; guodan@hfut.edu.cn).

Weili Wu and Dingzhu Du are with the Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: weiliwu@utdallas.edu; dzdu@utdallas.edu).

Digital Object Identifier 10.1109/TNET.2020.2993945

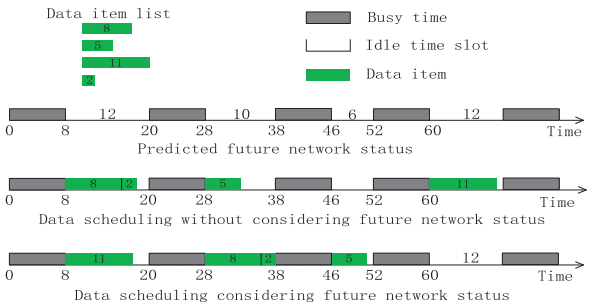


Fig. 1. Example of data transmission scheduling considering future network status.

the four data items in the second, third, first, and second time slots, respectively, which makes the makespan of shuffle scheduling be 51. Obviously, we can reduce the makespan of shuffle scheduling by predicting and exploiting the future network status.

It is reported that a large fraction of production jobs in data centers are recurring with predictable characteristics so that the data center network has predictable periodic busy and idle time, and the future job characteristics (e.g., execution time) of such jobs can be predicted with a high precision [10], [11]. With the emerging technology of software defined networking (SDN), SDN-enabled data centers can facilitate the prediction of the busy and idle time slots of the network by obtaining the network status such as network load, link bandwidth, etc [12]. However, to the best of our knowledge, limited research on the data transmission scheduling in MapReduce shuffle phase considers the predictable future networks status. This paper tackles the problem of shuffle scheduling for MapReduce jobs with the objective to minimize the makespan of shuffle phase by leveraging the periodic busy and idle states of the data center network.

The main contributions of this paper are as follows:

- We investigate the shuffle scheduling problem to reduce the makespan of MapReduce shuffle phase by exploiting the predictable future periodic network status.
- We formulate the shuffle scheduling problem and propose a simple yet effective network-aware shuffle scheduling algorithm (NAS) to reduce the number of idle time slots required to transfer the shuffle data so as to reduce the shuffle makespan with predictable future network status available. We prove that the proposed algorithm NAS is a $\frac{3}{2}$-approximation algorithm to the shuffle scheduling problem when all the future idle time slots have the same duration.
- We conduct simulations to evaluate the performance of the proposed algorithm. Simulation results demonstrate the proposed algorithm can effectively reduce the makespan of MapReduce shuffle phase and increase network utilization.

The rest of the paper is organized as follows. The related work is introduced in Section II. The problem is defined in Section III. The proposed algorithm and the analysis of the algorithm are presented in Section IV. The performance evaluation of the proposed algorithm is given in Section V, and the conclusions are detailed in Section VI.

II. Related Work

The transmission of intermediate data from mapper nodes to reducer nodes, i.e. shuffle, introduces a large amount of network traffic during the execution of MapReduce jobs. Optimizing data transmission scheduling in the shuffle phase helps MapReduce improve the job performance. Some research has been conducted on optimizing the data transmission scheduling in MapReduce.

Completion time is an important metric for flow scheduling. 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 [13]. A priority-based flow scheduling algorithm (PFO) was constructed for online social network (OSN) data center to decrease the average completion time for bursty flows and ensure a high throughput, where PFO divided the flows based on different metrics and allocated different rates to different flows on the basis of flow size and deadline information [14].

Some research aims to minimize the completion time for coflow. A heuristic algorithm called Fastest-Volume-Disposal-First was proposed to implement flow-level traffic compression and scheduling system (Swallow) by using traffic compression to reduce the amount of data transmitted over the network with the objective of minimizing coflow completion time (CCT) [15]. A multiple-attributes-based coflow scheduling (MCS) mechanism was presented 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 [9].

Some research focuses on scheduling the data transmission to meet deadline. A soft real-time transport protocol with deadline constraint in data centers was designed and a flow-based deadline scheduling scheme for data center networks (FBDS) was proposed to improve the deadline meeting rate [16]. A deadline-aware flow scheduling (DAFS) was proposed to decrease the deadline mismatch and blocking probabilities, thereby improving the average application throughput [17].

Some algorithms have been proposed to strike a tradeoff between the latency and the throughput of the network. A hybrid network architecture (HybridPass) for data center networks was introduced with the objective to support both time-triggered and event-triggered scheduling with respect to latency-sensitive and throughput-intensive flows; HybridPass introduced an arbiter which uses a loosely synchronized time-triggered manner to allocate the network bandwidth for latency-sensitive and throughput-intensive flows from a global perspective [18]. A flow scheduling scheme (Freeway) adaptively partitioned the available paths into low latency and high

throughput paths, and then provided different transmission services for each category; Freeway proposed a dynamic path partitioning algorithm to adjust dynamically the number of low latency and high throughput paths, where mice flows were transmitted over low latency paths using simple equal cost multiple path (ECMP) scheduling and elephant flows were sent on different high-throughput paths [19].

Some research combines SDN and Hadoop to improve MapReduce performance by exploiting the flexible network management capability provided by SDN. Cormorant combined flow scheduling algorithm (Hedera) and task placement algorithm (Mantri) to optimize MapReduce-based query processing system with SDN, by reducing the overhead caused by network congestion [20]. An SDN app for Hadoop clusters was proposed to control the network traffic to effectively alleviate network congestion during the shuffle phase by combining SDN and Hadoop, thereby improving MapReduce execution speed [21]. A dynamic network resource cooperation scheduling method was proposed to allocate network bandwidth and reduce data flow conflict with distributed SDN [22]. A responsive multipath TCP algorithm was proposed in SDN-based datacenters to improve the throughput of MapReduce with the SDN features [23]. A dynamic network scheduler was designed to provide different scheduling strategies for different network flows to reduce bandwidth competition, balance the workload of network links, increase bandwidth utilization, and utilize the OpenFlow to adjust transfers of flows dynamically [8]. An application-aware SDN routing algorithm was designed to optimize the total completion time of communication between servers by combining SDN and Hadoop [24]. A centralized flow-scheduling framework called Phurti was implemented and a heuristic was proposed with the goal of improving the completion time of jobs in a cluster shared by multiple tenants, by using OpenFlow collecting application and network information [25]. An SDN-based online scheduling framework was proposed to combine task scheduling and flow scheduling, and the framework selects the task placement based on the available bandwidth on the SDN switches and meanwhile optimally allocates the bandwidth to each data flow with the goal of increasing bandwidth utilization and reducing job completion time [26].

It is reported that a large fraction of production jobs in data centers are recurring with predictable characteristics, so the data center network has predictable periodic busy and idle time [10], [11]. However, limited research on the data flow scheduling in shuffle phase considers the predictable future networks status.

III. Problem Formulation

The symbols and notations used in the paper are listed in Table I. A large amount of data are transferred from mapper nodes to reducer nodes during the shuffle phase. Each mapper node transmits data to multiple reducer nodes, and each reducer node receives data from multiple mapper nodes. If there are data to be transmitted from mapper node m to reducer node r, these two nodes constitute a node pair (m, r). Multiple data items are transmitted between node pair (m, r), and the data are transmitted on path $p_{m,r}$ which is the shortest path between mapper node m and reducer node r by default. If the transmission time of a data item overlaps with that of another data item, we say these two data items collide with each other; otherwise, we say they are compatible with each other. If two paths share one or multiple links, we say these two paths conflict with each other; otherwise, we say they are compatible with each other. The data items colliding with each other cannot be transmitted on the conflicting paths.

TABLE I
TABLE OF NOTATIONS

m	m-th mapper node
r	r-th reducer node
M	Set of mapper nodes
R	Set of reducer nodes
D_m	List of the intermediate data items produced by the tasks assigned to mapper m.
$D_{m,r}$	List of the intermediate data items to be transferred from mapper m to reducer r
$d^n_{m,r}$	n-th data item in set $D_{m,r}$
$s^n_{m,r}$	Size of data $d^n_{m,r}$
$p_{m,r}$	Transmission path from mapper m to reducer r
$C_{m,r}$	Transmission capacity of path $p_{m,r}$
$L_{m,r}$	List of idle time slots on path $p_{m,r}$
$l^k_{m,r}$	k-th idle time slot in $L_{m,r}$
$\sigma(l^k_{m,r})$	Number of data items transferred in $l^k_{m,r}$
$f(l^k_{m,r})$	Time used for data transmission in time slot $l^k_{m,r}$
$t^k_{m,r}$	Duration time of slot $l^k_{m,r}$
$d^{k,n}_{m,r}$	n-th data item in set $D_{m,r}$ transferred in idle time slot $l^k_{m,r}$
$s(d^{k,n}_{m,r})$	Transmission time of data $d^{k,n}_{m,r}$
$E(d^{k,1}_{m,r})$	Set of data items other than $d^{k,1}_{m,r}$ in $l^k_{m,r}$
$x^{k,n}_{m,r}$	Indicates whether data $d^n_{m,r}$ is transferred in idle time slot $l^k_{m,r}$(=1) or not(=0)
$\tau^n_{m,r}$	Transmission time of data $d^n_{m,r}$ on path $p_{m,r}$
$\tau^{s,k}_{m,r}$	Start time of idle time slot $l^k_{m,r}$
$\tau^{e,k}_{m,r}$	End time of idle time slot $l^k_{m,r}$
$\tau^e_{m,r}$	Transmission completion time of all the data between mapper m and reducer r

The data can be transfered when the transmission path is idle. We can schedule the data transmission in shuffle phase efficiently, if the busy and idle time of the network is available, which is possible in SDN-enabled data center networks. A large number of business-critical jobs are recurring with pre-defined submission time and predictable resource requirements, and hence we can predict the network status with the network control capability provisioned by SDN.

In this paper, we aim to find an efficient strategy to schedule the data transmission during the shuffle phase. That is, for a MapReduce job, given a set of node pairs and the data to be transferred during the shuffle phase, our objective is to minimize the makespan of shuffle phase with the future predictable periodic network status available. In other words, our objective is to

$$\text{Minimize} \quad \max_{m\in M, r\in R}\{\tau^e_{m,r}\} \tag{1}$$

$$\text{Subject to:} \ \tau^n_{m,r} = \frac{s^n_{m,r}}{C_{m,r}}, \quad \forall m \in M,\ \forall r \in R,\ \forall n \in D_{m,r} \tag{2}$$

$$\tau^{s,k}_{m,r} + x^{k,n}_{m,r} \cdot \tau^n_{m,r} \le \tau^{e,k}_{m,r}, \forall m \in M, \forall r \in R, \forall n \in D_{m,r},\ \forall k \in L_{m,r} \tag{3}$$

$$\tau^e_{m,r} = \max_{k \in L_{m,r}} \{\tau^{s,k}_{m,r} + \sum_{n=1}^{|D_{m,r}|} \left(x^{k,n}_{m,r} \cdot \tau^n_{m,r}\right)\}, \quad \forall m \in M, \forall r \in R \tag{4}$$

$$\sum_{k=1}^{|L_{m,r}|} x^{k,n}_{m,r} = 1, \quad \forall m \in M, \ \forall r \in R, \ \forall d^n_{m,r} \in D_{m,r} \tag{5}$$

$$x^{k,n}_{m,r} \in \{0, 1\} \tag{6}$$

Eq. (2) defines the transmission time of the intermediate data. Eq. (3) ensures that each data item should finish transmission in an idle time slot. Eq. (4) determines the transmission finish time of data item list $D_{m,r}$. Eq. (5) signifies that each data item is transmitted once. Eq. (6) mandates that each $x^{k,n}_{m,r}$ is a binary variable.

IV. Network-Aware Shuffle Scheduling Algorithm for MapReduce Jobs

The recurring jobs split the network into periodic busy and idle time slots, and we need to schedule all the data into different idle time slots. Each time slot can be used to transmit multiple data items. The problem of scheduling data in one time slot is similar to the bin packing problem which is an $\mathcal{NP}$-complete problem. Our problem needs to schedule the data in multiple time slots to minimize the makespan, and obviously our problem is also $\mathcal{NP}$-complete.

In this section, we propose a Network-Aware Shuffle scheduling algorithm (NAS) to schedule the data in the shuffle phase for MapReduce jobs with the future predictable periodic network status available. Two data items colliding with each other cannot be transmitted on the conflicting paths. When conflict happens between the paths of two node pairs, we need to find an alternative path for one of the node pairs. Different node pairs have different amount of data to be transferred. We sort the node pairs by the amount of data between the node pairs so that the data are scheduled in sequence. The data are transmitted in different time slots on different paths such that the makespan of shuffle phase is minimized. Algorithm NAS consists of three stages: (1) node pairs and data sorting, (2) conflict reduction, and (3) data scheduling.

A. Node Pairs and Data Sorting

The sorting algorithm sorts the node pairs and the data between each node pair. Each node pair may transmit multiple data items. The node pairs are sorted according to the non-ascending order of the amount of data to be transmitted between the node pairs. If multiple node pairs need to transmit the same amount of data, the sorting algorithm sorts the node pairs by the non-ascending order of the path lengths between node pairs. After sorting the node pairs, the sorting algorithm sorts the data between each node pair according to the non-ascending order of data sizes.

B. Conflict Reduction

The algorithm shown in Algorithm 1 reduces the conflicts among the paths by changing the paths of node pairs, if two node pairs have conflict paths. When the idle time lists of two node pairs (m_a, r_a) and (m_b, r_b) overlap with each other, we determine which node pair needs to change the path. Let the node pair to change the path be $(m', r') \in \{(m_a, r_a), (m_b, r_b)\}$. We select the node pair with fewer data to change the transmission path so that more data can go through the shortest path. If the two node pairs need to transfer the same amount of data, the node pair with the shorter path length will change the transmission path, such that the path change may potentially affect fewer other node pairs. If the conflict cannot be avoided by changing the transmission path, we use the shortest path as the transmission path between the node pair which waits to transfer the data until the path is idle.

Algorithm 1 Conflict Reduction

Input: Two node pairs of (m_a, r_a) and (m_b, r_b) conflict in transmission paths; idle time lists L_{m_a,r_a} and L_{m_b,r_b}.
Output: The updated transmission path of the node pair which needs to change the transmission path; the idle time lists of the updated transmission path.

```
if L_{m_b,r_b} ∩ L_{m_a,r_a} = ∅ then
    return;
end if
if |D_{m_a,r_a}| > |D_{m_b,r_b}| then
    (m', r') = (m_b, r_b)
else if |D_{m_a,r_a}| < |D_{m_b,r_b}| then
    (m', r') = (m_a, r_a)
else
    if the path length of (m_a, r_a) ≥ the path length of (m_b, r_b) then
        (m', r') = (m_b, r_b)
    else
        (m', r') = (m_a, r_a)
    end if
end if
Set the conflicting links as invalid links;
Recalculate the shortest path for (m', r') using Dijkstra's algorithm as the alternative transmission path and get the idle time X on the alternative transmission path;
if the transmission path exists then
    Update the transmission path of (m', r');
end if
return the updated transmission paths, X.
```

C. Data Scheduling

After reducing the path conflicts, the data of each node pair are scheduled in the idle time slots as shown in Algorithm 2. Assume that the scheduling process will use $SlotNum$ time slots to transmit the data in list $D_{m,r}$. The algorithm proceeds iteratively. Within each iteration, we try to fully utilize each time slot by putting the biggest data and the smallest data in terms of data sizes among all the data in the time slot. For time slot $l^k_{m,r}$, we check whether the biggest data can be transferred in the slot. If yes, we check wether the smallest data can be transferred in slot $l^k_{m,r}$. If yes, we try the second biggest data. If it can also be put in slot $l^k_{m,r}$, the second smallest data will be tried. If the biggest data cannot be transmitted in slot $l^k_{m,r}$, we try to use the next time slot $l^{k+1}_{m,r}$. The scheduling repeats

Algorithm 2 Data Scheduling

Input: Data list $D_{m,r}$, idle time slot list $L_{m,r}$.

Output: The transmission completion time of $D_{m,r}$; number of idle time slots used to transfer $D_{m,r}$.

1: Initialize $left = 1$, $right = |D_{m,r}|$, $k = 1$, $Flag = false$;
2: **while** $left \leq right$ **do**
3: **if** $\tau_{m,r}^{s,k} + \frac{s_{m,r}^{left}}{C_{m,r}} \leq \tau_{m,r}^{e,k}$ **then**
4: Use $l_{m,r}^{k}$ to transfer $d_{m,r}^{left}$;
5: $\tau_{m,r}^{s,k} = \tau_{m,r}^{s,k} + \frac{s_{m,r}^{left}}{C_{m,r}}$;
6: $left = left + 1$, $Flag = true$;
7: **else if** $Flag = false$ **then**
8: $k = k + 1$;
9: **end if**
10: **if** $\tau_{m,r}^{s,k} + \frac{s_{m,r}^{right}}{C_{m,r}} \leq \tau_{m,r}^{e,k}$ and $Flag$ **then**
11: Use $l_{m,r}^{k}$ to transfer $d_{m,r}^{right}$;
12: $\tau_{m,r}^{s,k} = \tau_{m,r}^{s,k} + \frac{s_{m,r}^{right}}{C_{m,r}}$;
13: $right = right - 1$;
14: **else if** $Flag$ **then**
15: $k = k + 1$, $Flag = false$;
16: **end if**
17: **end while**
18: $SlotNum = k$;
19: **return** $\tau_{m,r}^{s,SlotNum}$, $SlotNum$.

the process of putting big and small data items in the time slot. When slot $l_{m,r}^{k}$ cannot accommodate the big data any more, we continue to try the small data until the time slot cannot transfer any more data items.

D. Algorithm Proof

When all the idle time slots have the same duration, we prove our algorithm is a $\frac{3}{2}$-approximation algorithm to the shuffle scheduling problem.

Lemma 1: Assume $SlotNum$ is the number of idle time slots returned by Algorithm 2. If there are at least $SlotNum-1$ idle time slots and anyone of these $SlotNum-1$ idle time slots transfers no less than 2 data items, e.g. $\sigma(l_{m,r}^{k}) \geq 2$, each of these $SlotNum-1$ idle time slots takes more than $\frac{2}{3}$ of the slot duration time to transfer data.

Proof: Assuming that there are two idle time slots $l_{m,r}^{i}$ and $l_{m,r}^{j}$, let the data transferred in $l_{m,r}^{i}$ be $d_{m,r}^{i,1}$, $d_{m,r}^{i,2}$, ..., $d_{m,r}^{i,\sigma(l_{m,r}^{i})}$, and the data transferred in $l_{m,r}^{j}$ be $d_{m,r}^{j,1}$, $d_{m,r}^{j,2}$, ..., $d_{m,r}^{j,\sigma(l_{m,r}^{j})}$. $d_{m,r}^{i,n}$ and $d_{m,r}^{j,n'}$ are the n-th data transferred in slot $l_{m,r}^{i}$ and the n'-th data item transferred in slot $l_{m,r}^{j}$, respectively, where $n \in \{1,2,\ldots,\sigma(l_{m,r}^{i})\}$, $n' \in \{1,2,\ldots,\sigma(l_{m,r}^{j})\}$. According to Algorithm 2, data $d_{m,r}^{i,\sigma(l_{m,r}^{i})}$ and $d_{m,r}^{j,\sigma(l_{m,r}^{j})}$ are the last data transferred in slots $l_{m,r}^{i}$ and $l_{m,r}^{j}$, respectively. Both of these two slots take less than $\frac{2}{3}$ of the duration time to transfer data; that is, $f(l_{m,r}^{i}) \leq \frac{2}{3} \cdot t_{m,r}^{i}$, $f(l_{m,r}^{j}) \leq \frac{2}{3} \cdot t_{m,r}^{j}$ $(1 \leq i < j \leq SlotNum)$.

According to the assumption above, we can get

$$f(l_{m,r}^{i}) = s(d_{m,r}^{i,1}) + s(d_{m,r}^{i,2}) + \ldots + s(d_{m,r}^{i,\sigma(l_{m,r}^{i})}) \leq \frac{2}{3} \cdot t_{m,r}^{i} \tag{7}$$

$$f(l_{m,r}^{j}) = s(d_{m,r}^{j,1}) + s(d_{m,r}^{j,2}) + \ldots + s(d_{m,r}^{j,\sigma(l_{m,r}^{j})}) \leq \frac{2}{3} \cdot t_{m,r}^{j} \tag{8}$$

According to Algorithm 2, Eqs. (9) and (10) are established

$$s(d_{m,r}^{i,2}) \leq s(d_{m,r}^{i,\sigma(l_{m,r}^{i})}) \leq s(d_{m,r}^{i,1}) \tag{9}$$

$$s(d_{m,r}^{j,2}) \leq s(d_{m,r}^{j,\sigma(l_{m,r}^{j})}) \leq s(d_{m,r}^{j,1}) \tag{10}$$

Because $i < j$, we have $s(d_{m,r}^{i,1}) \geq s(d_{m,r}^{j,1})$ and $t_{m,r}^{i} \geq t_{m,r}^{j}$. Assuming that data $d_{m,r}^{i+1,2}$ is the second data item transferred in $l_{m,r}^{i+1}$, according to Algorithm 2, we can know that

$$f(l_{m,r}^{i}) + s(d_{m,r}^{i+1,2}) > t_{m,r}^{i} \tag{11}$$

Otherwise, data $d_{m,r}^{i+1,2}$ can be transferred in $l_{m,r}^{i}$.

We discuss two different cases as follows:

Case 1: when $s(d_{m,r}^{j,1}) \geq \frac{1}{3} \cdot t_{m,r}^{j}$, there must be $\frac{2}{3} \cdot t_{m,r}^{i} \geq s(d_{m,r}^{i,1}) \geq s(d_{m,r}^{j,1}) \geq \frac{1}{3} \cdot t_{m,r}^{j}$. According to Eqs. (7) and (8), we can get $f(l_{m,r}^{i}) + f(l_{m,r}^{j}) \leq \frac{2}{3} \cdot (t_{m,r}^{i} + t_{m,r}^{j})$, which means $s(d_{m,r}^{i,1}) + s(d_{m,r}^{i,2}) + \ldots + s(d_{m,r}^{i,\sigma(l_{m,r}^{i})}) + s(d_{m,r}^{j,1}) + s(d_{m,r}^{j,2}) + \ldots + s(d_{m,r}^{j,\sigma(l_{m,r}^{j})}) \leq \frac{2}{3} \cdot (t_{m,r}^{i} + t_{m,r}^{j})$, $s(d_{m,r}^{i,1}) + s(d_{m,r}^{i,2}) + \ldots + s(d_{m,r}^{i,\sigma(l_{m,r}^{i})}) + s(d_{m,r}^{j,2}) + \ldots + s(d_{m,r}^{j,\sigma(l_{m,r}^{j})}) \leq \frac{2}{3} \cdot (t_{m,r}^{i} + t_{m,r}^{j}) - s(d_{m,r}^{j,1}) \leq \frac{2}{3} \cdot t_{m,r}^{i} + \frac{1}{3} \cdot t_{m,r}^{j} \leq t_{m,r}^{i}$, and $s(d_{m,r}^{i,1}) + s(d_{m,r}^{i,2}) + \ldots + s(d_{m,r}^{i,\sigma(l_{m,r}^{i})}) + s(d_{m,r}^{j,2}) \leq t_{m,r}^{i}$, so $f(l_{m,r}^{i}) + s(d_{m,r}^{j,2}) \leq t_{m,r}^{i}$.

Case 2: when $s(d_{m,r}^{j,1}) < \frac{1}{3} \cdot t_{m,r}^{j}$, according to Eq. (10), we can get $s(d_{m,r}^{j,2}) \leq s(d_{m,r}^{j,\sigma(l_{m,r}^{j})}) \leq s(d_{m,r}^{j,1}) < \frac{1}{3} \cdot t_{m,r}^{j} \leq \frac{1}{3} \cdot t_{m,r}^{i}$. Based on the assumption $f(l_{m,r}^{i}) \leq \frac{2}{3} \cdot t_{m,r}^{i}$, we can get $f(l_{m,r}^{i}) + s(d_{m,r}^{j,2}) \leq \frac{2}{3} \cdot t_{m,r}^{i} + \frac{1}{3} \cdot t_{m,r}^{i}$, and $f(l_{m,r}^{i}) + s(d_{m,r}^{j,2}) < t_{m,r}^{i}$.

According to Algorithm 2, we can get $s(d_{m,r}^{i+1,2}) \leq s(d_{m,r}^{j,2})$. Since $f(l_{m,r}^{i}) + s(d_{m,r}^{j,2}) < t_{m,r}^{i}$, we know $f(l_{m,r}^{i}) + s(d_{m,r}^{i+1,2}) < t_{m,r}^{i}$, which contradicts to Eq. (11). Therefore, if there are two such idle time slots $l_{m,r}^{i}$ and $l_{m,r}^{j}$, $1 \leq i < j \leq SlotNum$, making $f(l_{m,r}^{i}) \leq \frac{2}{3} \cdot t_{m,r}^{i}$ and $f(l_{m,r}^{j}) \leq \frac{2}{3} \cdot t_{m,r}^{j}$, the second data $d_{m,r}^{i+1,2}$ transferred in $l_{m,r}^{i+1}$ can also be transferred in $l_{m,r}^{i}$, which contradicts to Eq. (11) obtained by Algorithm 2. The lemma is proven. ■

Lemma 2: Assume $SlotNum$ is the number of idle time slots which is returned by Algorithm 2 used to transfer the data in list $D_{m,r}$ and the k-th idle time slot transfers $\sigma(l_{m,r}^{k})$ number of data items. If there are at least $SlotNum-1$ idle time slots and anyone of these $SlotNum-1$ idle time slots transfers no less than 2 data items, i.e $\sigma(l_{m,r}^{k}) \geq 2$, $(k \in 1,2,\ldots,SlotNum)$, Algorithm 2 is a $\frac{3}{2}$-approximation algorithm to the shuffle scheduling problem.

Proof: We assume the optimal number of time slots to schedule the data is OPT. $\exists m^{*} \in \{1,2,\ldots,SlotNum\}$, $\forall j \in \{1,2,\ldots,SlotNum\} \setminus \{m^{*}\}$, $\sum_{j=1}^{SlotNum} f(l_{m,r}^{j}) = \sum_{\substack{j=1 \\ j \neq m^{*}}}^{SlotNum} f(l_{m,r}^{j}) + f(l_{m,r}^{m^{*}})$.

Now, we claim $f(l_{m,r}^{m^{*}}) > t_{m,r}^{m^{*}} - f(l_{m,r}^{j})$, which means all the data transferred in any idle time slot of the other $SlotNum-1$ idle time slots cannot be transferred in $l_{m,r}^{m^{*}}$.

To prove the claim, we assume there is an idle time slot $l^h_{m,r}$ and all the data transferred in $l^h_{m,r}$ can be transferred in $l^{m^*}_{m,r}$, which means $f(l^{m^*}_{m,r}) + f(l^h_{m,r}) \leq t^{m^*}_{m,r}$.

We discuss two different cases as follows:

Case 1: $h < m^*$. According to Algorithm 2, $s(d^{h+1,2}_{m,r}) \leq f(l^{m^*}_{m,r})$, and we can get $f(l^h_{m,r}) + s(d^{h+1,2}_{m,r}) \leq f(l^h_{m,r}) + f(l^{m^*}_{m,r}) \leq t^{m^*}_{m,r} \leq t^h_{m,r}$. That is, data $d^{h+1,2}_{m,r}$ transferred in idle time slot $l^{h+1}_{m,r}$ can also be transferred in $l^h_{m,r}$, which contradicts to $f(l^h_{m,r}) + s(d^{h+1,2}_{m,r}) > t^h_{m,r}$ obtained by Algorithm 2.

Case 2: $m^* < h$. Because $f(l^{m^*}_{m,r}) + f(l^h_{m,r}) \leq t^{m^*}_{m,r}$ and $s(d^{m^*+1,2}_{m,r}) \leq f(l^h_{m,r})$, we can know $f(l^{m^*}_{m,r}) + s(d^{m^*+1,2}_{m,r}) \leq t^{m^*}_{m,r}$. That is, data $d^{m^*+1,2}_{m,r}$ transferred in idle time slot $l^{m^*+1}_{m,r}$ can also be transferred in $l^{m^*}_{m,r}$, which contradicts to $f(l^{m^*}_{m,r}) + s(d^{m^*+1,2}_{m,r}) > t^{m^*}_{m,r}$ obtained by Algorithm 2. The claim is proven.

With the claim above, we can get $f(l^{m^*}_{m,r}) > \max\{t^{m^*}_{m,r} - f(l^i_{m,r})\} \geq \frac{1}{SlotNum-1} \cdot \sum_{\substack{i=1 \\ i\neq m^*}}^{SlotNum} (t^{m^*}_{m,r} - f(l^i_{m,r})) = t^{m^*}_{m,r} - \frac{1}{SlotNum-1} \cdot \sum_{\substack{i=1 \\ i\neq m^*}}^{SlotNum} (f(l^i_{m,r})),\ i = 1, 2, \ldots, SlotNum, i \neq m^*$, so $f(l^{m^*}_{m,r}) - \frac{1}{SlotNum-1} \cdot f(l^{m^*}_{m,r}) > t^{m^*}_{m,r} - \frac{1}{SlotNum-1} \cdot \sum_{i=1}^{SlotNum} (f(l^i_{m,r})) > t^{m^*}_{m,r} - \frac{1}{SlotNum-1} \cdot OPT \cdot t^1_{m,r}$, and then $\frac{SlotNum-2}{SlotNum-1} \cdot f(l^{m^*}_{m,r}) > t^{m^*}_{m,r} - \frac{1}{SlotNum-1} \cdot OPT \cdot t^1_{m,r}$, $f(l^{m^*}_{m,r}) > \frac{SlotNum-1}{SlotNum-2} \cdot t^{m^*}_{m,r} - \frac{1}{SlotNum-2} \cdot OPT \cdot t^1_{m,r}$. According to Algorithm 2, $OPT \cdot t^1_{m,r} \geq f(l^{m^*}_{m,r}) + \sum_{\substack{i=1 \\ i\neq m^*}}^{SlotNum} (f(l^i_{m,r})) > f(l^{m^*}_{m,r}) + \frac{2}{3} \cdot (SlotNum - 1) \cdot t^{SlotNum-1}_{m,r} > \frac{SlotNum-1}{SlotNum-2} \cdot t^{m^*}_{m,r} - \frac{1}{SlotNum-2} \cdot OPT \cdot t^1_{m,r} + \frac{2}{3} \cdot (SlotNum-1) \cdot t^{SlotNum-1}_{m,r}$, and we get $\frac{SlotNum-1}{SlotNum-2} OPT \cdot t^1_{m,r} > \frac{SlotNum-1}{SlotNum-2} \cdot t^{m^*}_{m,r} + \frac{2}{3} \cdot (SlotNum - 1) \cdot t^{SlotNum-1}_{m,r}$, $OPT \cdot t^1_{m,r} > t^{m^*}_{m,r} + \frac{2(SlotNum-2)}{3} \cdot t^{SlotNum-1}_{m,r}$, $OPT \cdot t^1_{m,r} > \frac{2SlotNum-1}{3} \cdot t^{SlotNum-1}_{m,r}$. Because $SlotNum$ and OPT are integers, we get $3 \cdot OPT \geq 2 \cdot SlotNum$, which means $\frac{SlotNum}{OPT} \leq \frac{3}{2}$. The lemma is proven. ■

Lemma 3: Assume $SlotNum$ is the number of idle time slots which is returned by Algorithm 2 used to transfer the data in list $D_{m,r}$ and the k-th idle time slot transfers $\sigma(l^k_{m,r})$ number of data items. If there are num idle time slots, $2 \leq num \leq SlotNum$, each of the num idle time slots transfers only one data item, and all the data items transferred in the num idle time slots constitute a set UN. There must be an optimal solution OPT, such that there are also num idle time slots, each of which transfers only one data item and the data set the num slots transfer equals UN.

Proof: Assume that the scheduling results, including $SlotNum$ and the data transferring time returned by Algorithm 2, constitute the solution I and the optimal solution is I^*. $l^{k'}_{m,r}$ is an arbitrary idle time slot of the num idle time slots and $l^{k'}_{m,r}$ is also the j-th idle time slot returned by Algorithm 2. The only data transferred in $l^{k'}_{m,r}$ is data $d^{j,1}_{m,r}$.

Assume in I^*, there is an idle time slot $l^{j^*}_{m,r}$ which transfers not only data $d^{j,1}_{m,r}$ but also a data set $E(d^{j,1}_{m,r})$ consisting of some other data. Note the data in $E(d^{j,1}_{m,r})$ is smaller than $d^{j,1}_{m,r}$, and hence the data in $E(d^{j,1}_{m,r})$ must be some of the data which have been transferred in $l^i_{m,r}$, $(i < j)$ in I and data $d^{i,1}_{m,r}$ is not in $E(d^{j,1}_{m,r})$. If $d^q_{m,r}$ is the first data that cannot be transferred in $l^j_{m,r}$ according to Algorithm 2, we get $E(d^{j,1}_{m,r}) \subset \{d^{q+1}_{m,r}, d^{q+2}_{m,r}, \ldots\}$. Assume $l^{i^*}_{m,r}$ is the time slot which transfers $d^{i,1}_{m,r}$ in I^*. All the data in $E(d^{j,1}_{m,r})$ can also be transferred in $l^{i^*}_{m,r}$. As a result, we can convert solution I^* to another optimal solution I', in which there is an idle time slot transfers the only one data $d^{j,1}_{m,r}$. In other words, $l^{j^*}_{m,r}$ transfers only one data item, i.e. data $d^{j,1}_{m,r}$. The lemma is proven. ■

Theorem 1: Assume $SlotNum$ is the number of idle time slots used to transfer data list $D_{m,r}$, which is returned by Algorithm 2. If there are num idle time slots, each of which transfers only one data, Algorithm 2 is a $\frac{3}{2}$-approximation algorithm to the shuffle scheduling problem.

Proof: When $num = 1$, according to Lemma 2, the theorem is proven.

When $2 \leq num \leq SlotNum$, according to Lemma 3, assume the optimal number of time slots used to transfer the data is OPT, and the OPT time slots consist of two parts: one is the num idle time slots each of which transfers only one data item, and the other is OPT' time slots each of which transfers no less than two data items. All the data transferred in num idle time slots constitute set UN. The $SlotNum$ idle time slots returned by Algorithm 2 also consist of two parts: one is the num time slots which transfer the same set UN with only one data item in each slot, and the other is num' time slots with each transferring no less than two data items. We can get $OPT = OPT' + num$, $SlotNum = num' + num$. According to Lemma 2, $\frac{num'}{OPT'} \leq \frac{3}{2}$, we get $2 \cdot num' \leq 3 \cdot OPT'$, and $\frac{SlotNum}{OPT} = \frac{2 \cdot (num + num')}{2 \cdot (num + OPT')} \leq \frac{2 \cdot num + 3 \cdot OPT'}{2 \cdot (num + OPT')} \leq \frac{3 \cdot (num + OPT')}{2 \cdot (num + OPT')} = \frac{3}{2}$. That is, $\frac{SlotNum}{OPT} \leq \frac{3}{2}$. The theorem is proven. ■

E. Time Complexity

Assume that N is the maximum number of data between node pairs, V is the number of nodes in the network, and P is the number of node pairs. In the node pairs and data sorting stage, the sorting algorithm can be mergesort or quicksort. It takes $O(Plog(P))$ time to sort the node pairs. In the case that multiple node pairs transmit the same amount of data, the algorithm needs at most another $O(Plog(P))$ time to sort the node pairs by the path lengths. It takes $Nlog(N)$ time to sort the data for each node pair, so it consumes $PNlog(N)$ time in data sorting for all the node pairs. Therefore, the sorting of node pairs and data in stage 1 can be performed in $O(2Plog(P) + PNlog(N))$ time. We use Dijkstra's algorithm to find the shortest path between each node pair, and Dijkstra's algorithm runs in $O(V^2)$ time. In the worst case during the conflict reduction stage, all the node pairs conflict with each other in the transmission paths, and it takes at most $O(\frac{P(P+1)}{2} V^2)$ time to do the conflict reduction. In the data scheduling stage, we need to allocate the data between the P node pairs to the idle time slots. Each node pair transmits at most N data, and each node pair takes $O(N)$ time to

transmit the N data, since Algorithm 2 traverses each data item only once to decide the transmission time slot for the data item. Therefore, the data scheduling algorithm in stage 3 takes $O(PN)$ time. In summary, the time complexity of algorithm NAS is $O(2Plog(P) + PNlog(N) + \frac{P(P+1)}{2}V^2 + PN) = O(Plog(P) + PNlog(N) + \frac{P(P+1)}{2}V^2)$.

V. Performance Evaluation

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

A. Simulation Setup

We evaluate the proposed algorithm NAS in terms of makespan, MSRP (MakeSpan Reduction Percentage) [24], and network utilization against the optimal solution *OPT* [27], [28], algorithm *AAR* proposed for the most similar problem in [24], and the default greedy algorithm, denoted as *GSS*, in MapReduce [29]. The optimal solution is obtained by solving the shuffle scheduling problem in Section III using CPLEX 12.8.0. MSRP is the reduction percentage of the makespan comparing NAS with the two benchmark algorithms AAR and GSS. Algorithm AAR minimizes the shuffle time by sorting the data flows according to the path lengths between the node pairs, and choosing a path for each flow to ensure that the network is not blocked. Algorithm GSS sorts the node pairs and data in the same way as NAS. However, GSS does not consider the path conflicts and always finds the time slot for each data item which can finish the transmission as early as possible without the knowledge of the future network status.

We conduct the simulations under two scenarios: (1) the durations of future idle time slots are different, and (2) all the future idle time slots have the same duration. In the first scenario, the durations of idle time slots are randomly generated in the ranges of 0-50ms and 0-500ms, when the data sizes are within the ranges of 0-10MB and 0-100MB, respectively. In the second scenario, the duration of idle time slots is set as 10ms and 100ms, when the ranges of the data sizes are [0, 10]MB and [0, 100]MB, respectively. The network bandwidth for all the paths is set as $1GB/s$.

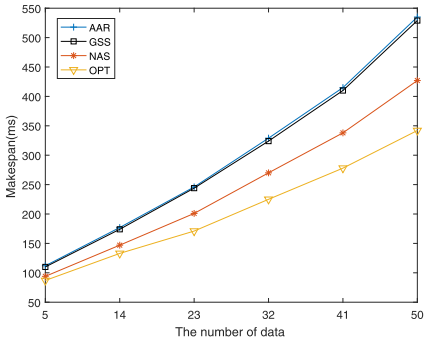
The network topology used in the simulation is a fat tree which is similar to that used in [24] and commonly used in data centers. With the fat tree, the path length between a node pair is as follows: if the two nodes are placed in the same rack, the path length is 2; if the two nodes are placed in different racks but in the same pod, the path length is 4; if the two nodes are placed in different pods, the path length is 6. The mapper and the reducer nodes are randomly chosen from the network. We conduct the simulations for 30 runs, and the average value of the running results is taken as the final result.

B. Performance Evaluation of the Proposed Algorithm

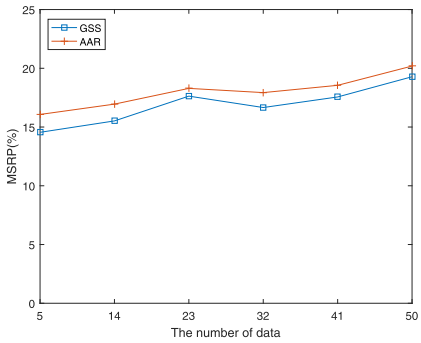
1) Impact of Various Number of Data: We evaluate the performances of different algorithms NAS, AAR, GSS and OPT by varying the number of data. Assuming there are 12 mapper nodes and 6 reducer nodes. The number of data transferred between each node pair increases from 5 to 50, which reflects the normal data volume between the node pairs [29]. To conduct the performance comparison under different data sizes, the sizes of data are randomly generated in 1-10MB and 1-100MB as shown in Figs. 2 and 3, respectively. Figs. 2(a), 2(b), 3(a), and 3(b) show the makespan and MSRP performances under the first scenario with different idle time slot sizes, while Figs. 2(c), 2(d), 3(c), and 3(d) depict the results under the second scenario with the same size of idle time slots.

Figs. 2(a), 2(c), 3(a), and 3(c) illustrate the makespan for different algorithms with different number of data. The makespan of all algorithms increases when the number of data increases, because the total network transmission in the network increases with the increasing number of data. As the number of data increases, NAS is increasingly better than AAR and GSS. When the durations of idle time slots are different, Figs. 2(b) and 3(b) demonstrate that algorithm NAS outperforms both AAR and GSS from about 15% to 20% with the number of data between each node pair increasing from 5 to 50. When the durations of idle time slots are the same and the number of data between each node pair increases from 5 to 50, Fig. 2(d) shows that the performance obtained by algorithm NAS is about 17% and 10% better than algorithms AAR and GSS, respectively, and Fig. 3(d) illustrates that algorithm NAS outperforms algorithms AAR and GSS about 18% and 14%, respectively. Algorithm NAS performs the closest to OPT among the three algorithms of NAS, AAR, and GSS. Algorithm NAS schedules the shuffle data by jointly considering path conflict, data sizes and network status, algorithm AAR optimizes the shuffle phase by choosing the shortest paths without consideration of path conflicts and data sizes, while algorithm GSS always finds the earliest transmission path among all the paths without any consideration of path conflicts and future network status. It can be observed that algorithm NAS performs considerably better than the approximation ratio of $\frac{3}{2}$. When the durations of idle time slots are different and the number of data is 50, algorithm NAS is worse than OPT by 22% and 26% with the data sizes in the ranges of 1-10MB and 1-100MB, respectively. When the durations of idle time slots are the same and the number of data is 50, algorithm NAS obtains worse results than OPT by 5% and 9% with the data size ranges of 1-10MB and 1-100MB, respectively.

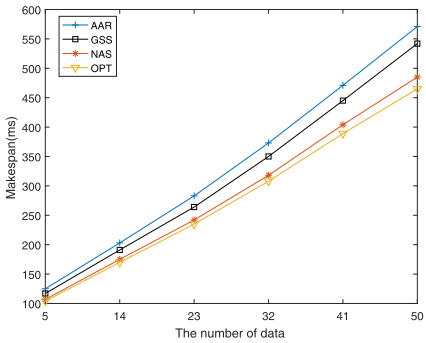
Figs. 2 and 3 demonstrate that the performance differences of the four algorithms are similar under different data sizes of 1-10MB and 1-100MB. Therefore, when evaluating the network utilization performances of the four algorithms, the data sizes are only randomly generated in 1-100MB with 12 mapper nodes and 6 reducer nodes. The number of data transferred between each node pair also increases from 5 to 50 as shown in Fig. 4. The network utilization of the four algorithms relatively keeps stable as the number of data increases. Fig. 4(a) shows that the network utilization of OPT, NAS, GSS, and AAR is about 99%, 88%, 82%, and 78%, respectively, with different idle time slot durations. Fig. 4(b) demonstrates that the network utilization of OPT, NAS, GSS, and AAR is approximately 95%, 93%, 87%,



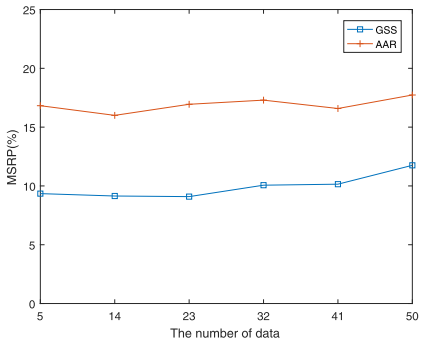
(a) Makespan with different idle time slot sizes



(b) MSRP with different idle time slot sizes

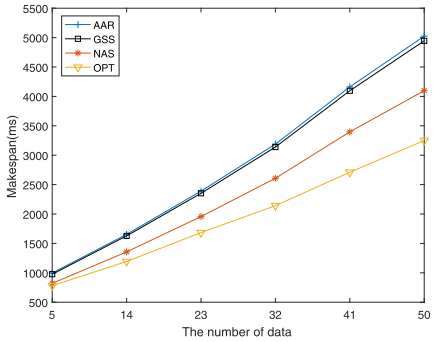


(c) Makespan with the same idle time slot size

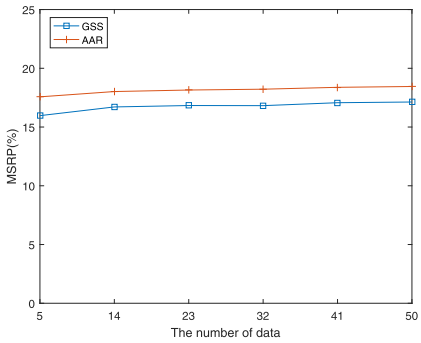


(d) MSRP with the same idle time slot sizes

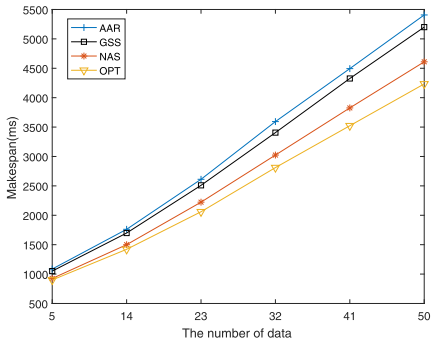
Fig. 2. The makespan and MSRP under different number of data with the sizes of data randomly generated in 1-10MB.



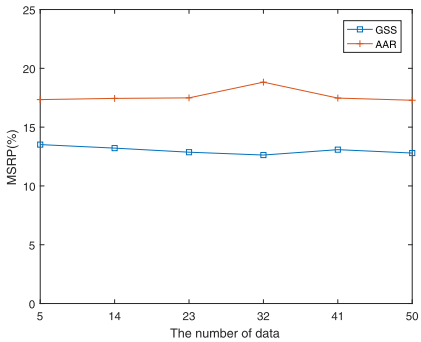
(a) Makespan with different idle time slot sizes



(b) MSRP with different idle time slot sizes



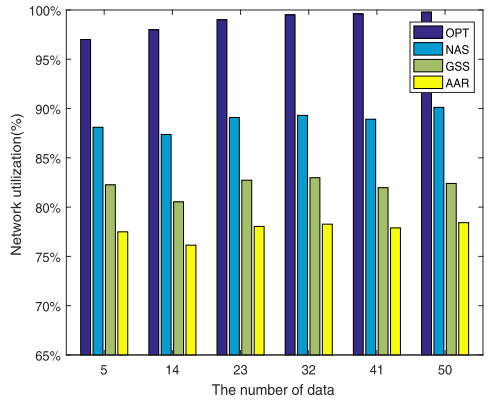
(c) Makespan with the same idle time slot size



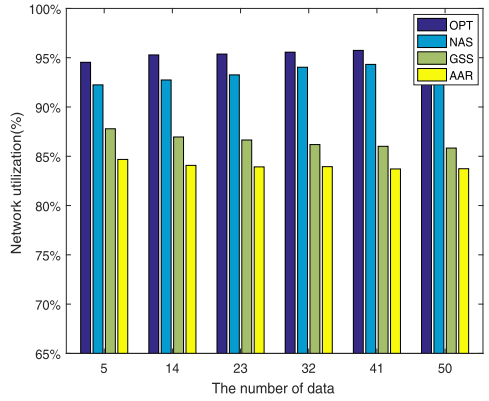
(d) MSRP with the same idle time slot size

Fig. 3. The makespan and MSRP under different number of data with the sizes of data randomly generated in 1-100MB.

and 84%, respectively, with the same duration of idle time slots. The network utilization of algorithm NAS is the closest to that of OPT, while AAR results in the lowest network utilization among the three algorithms of NAS, AAR, and GSS. Algorithm NAS schedules the data when network links are idle, thereby making good use of path idle time to schedule



(a) The performance with different idle time slot sizes

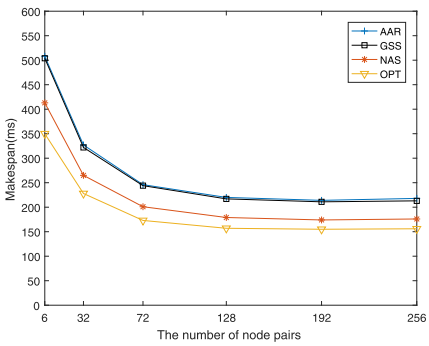


(b) The performance with the same idle time slot size

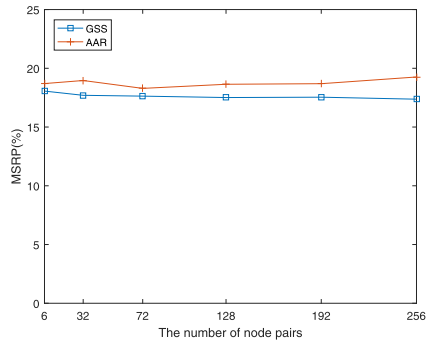
Fig. 4. The network utilization under different number of data with the sizes of data randomly generated in 1-100MB.

the data with the consideration of the predictable network status. Algorithm GSS schedules the data by finding the time slot for each data which can transmit the data as early as possible. Algorithm GSS is a greedy algorithm, and it cannot fully utilize the global network links. Algorithm AAR schedules the data by considering path lengths of node pairs instead of the network status. As a result, algorithm AAR potentially schedules the data on the busy links so that the data have to wait to be transmitted, and hence the network utilization is reduced.

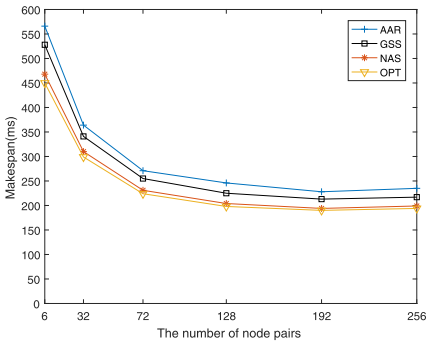
2) Impact of Various Number of Node Pairs: We study the impact of various number of node pairs on the performance of different algorithms. The number of node pairs increases from 6 (3 mapper nodes and 2 reducer nodes) to 256 (16 mapper nodes and 16 reducer nodes). We run the simulations when the number of node pairs are 6 (3 mapper nodes and 2 reducer nodes), 32 (8 mapper nodes and 4 reducer nodes), 72 (12 mapper nodes and 6 reducer nodes), 128 (16 mapper nodes and 8 reducer nodes), 192 (16 mapper nodes and 12 reducer nodes), and 256 (16 mapper nodes and 16 reducer nodes). Assuming the total number of data to be transferred is 1800 which are uniformly distributed across all the node pairs. To conduct the performance comparison under different data sizes, the sizes of data are randomly generated in 1-10MB and 1-100MB as shown in Fig. 5 and Fig. 6, respectively. Figs. 5(a), 5(b), 6(a), and 6(b) show the performance of makespan and MSRP under the first scenario with different



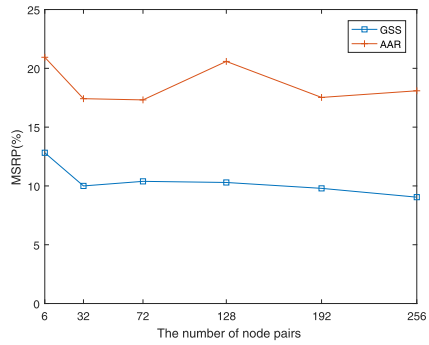
(a) Makespan with different idle time slot sizes



(b) MSRP with different idle time slot sizes

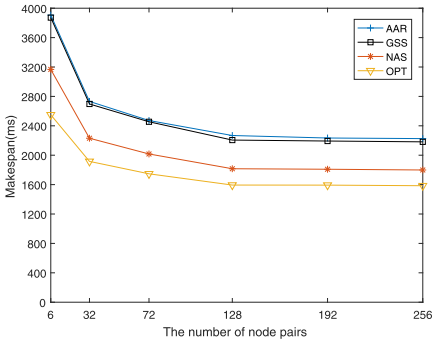


(c) Makespan with the same idle time slot size

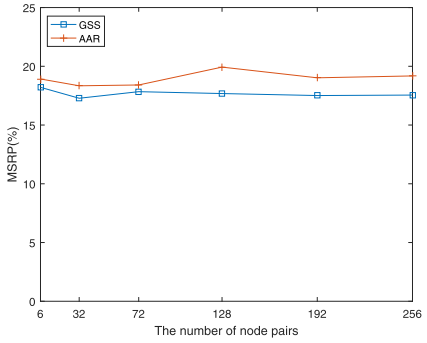


(d) MSRP with the same idle time slot size

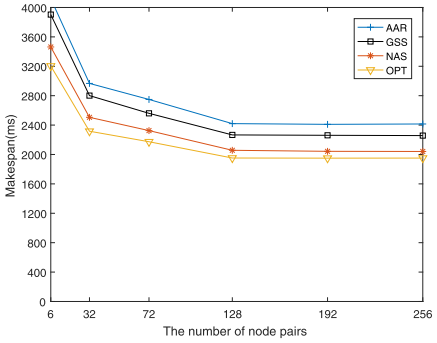
Fig. 5. The makespan and MSRP under different number of node pairs with the sizes of data randomly generated in 1-10MB.



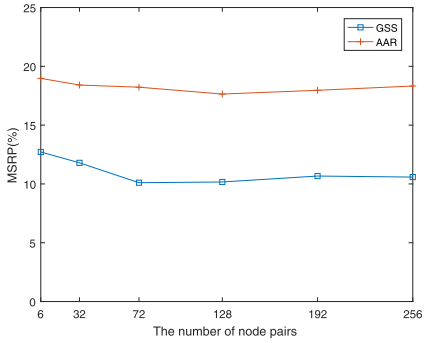
(a) Makespan with different idle time slot sizes



(b) MSRP with different idle time slot sizes



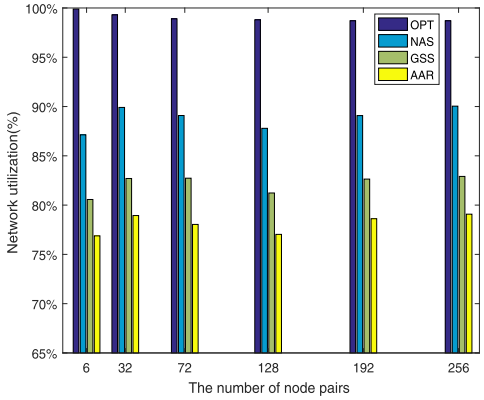
(c) Makespan with the same idle time slot size



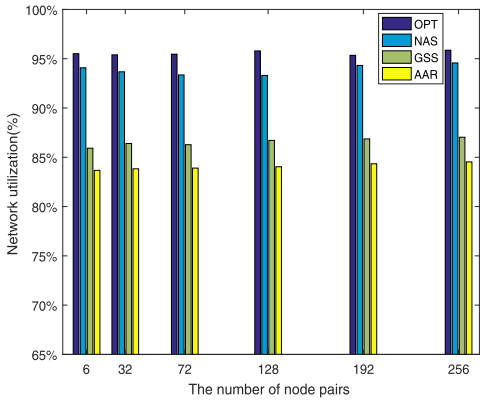
(d) MSRP with the same idle time slot size

Fig. 6. The makespan and MSRP under different number of node pairs with the sizes of data randomly generated in 1-100MB.

idle time slot durations, while Figs. 5(c), 5(d), 6(c), and 6(d) depict the results under the second scenario with the same size of idle time slots.



(a) The performance with different idle time slot sizes

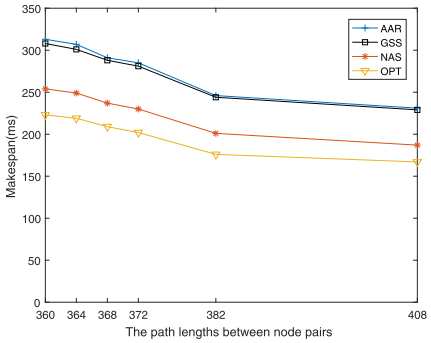


(b) The performance with the same idle time slot size

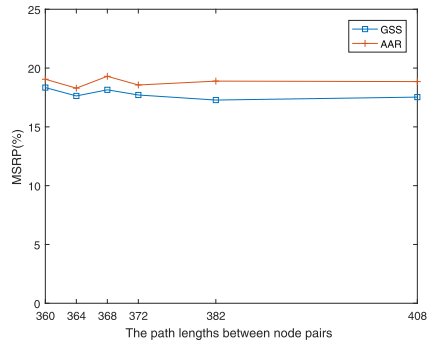
Fig. 7. The network utilization under different number of node pairs with the sizes of data randomly generated in 1-100MB.

Figs. 5(a), 5(c), 6(a) and 6(c) demonstrate the makespan performance by varying the number of node pairs. As the number of node pairs increases, the makespan of all the four algorithms decreases without regard to the data sizes. When the number of data is fixed, the increasing number of node pairs means the number of data each node pair needs to transfer decreases. The makespan keeps stable when the number of node pairs is bigger than 128, which indicates node pairs cannot reduce makespan when the number of node pairs increases. Figs. 5(b) and 6(b) show that algorithm NAS outperforms both algorithms AAR and GSS by about 20% when the durations of idle time slots are different. Algorithm NAS outperforms algorithms AAR and GSS by about 19% and 10%, respectively, with the same duration of idle time slots, as depicted in Figs. 5(d) and 6(d). Algorithm NAS obtains about 15% worse results than OPT with different idle time slot sizes, while achieving similar performance to OPT with the same size of idle time slots.

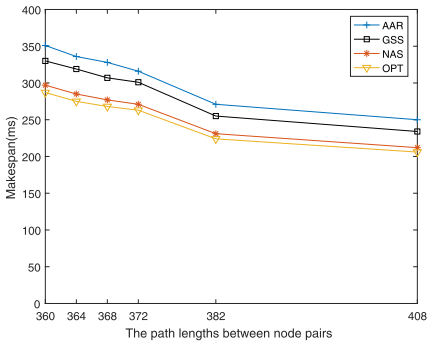
We evaluate the network utilization performance of the four algorithms by randomly generating the data with the data size range of $[1, 100]MB$. The number of node pairs also increases from 6 to 256. The total number of data to be transferred is 1800, and the data are uniformly distributed across all the node pairs. It can be observed from Fig. 7 that the network utilization of the four algorithms is relatively stable with the



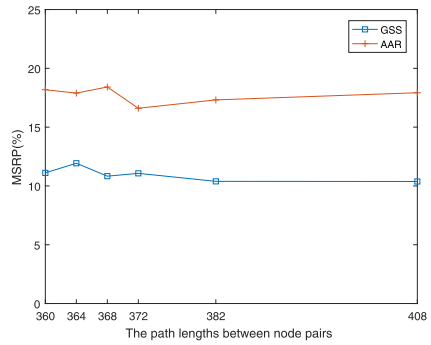
(a) Makespan with different idle time slot sizes



(b) MSRP with different idle time slot sizes

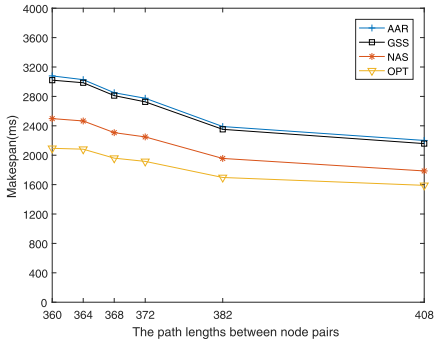


(c) Makespan with the same idle time slot size

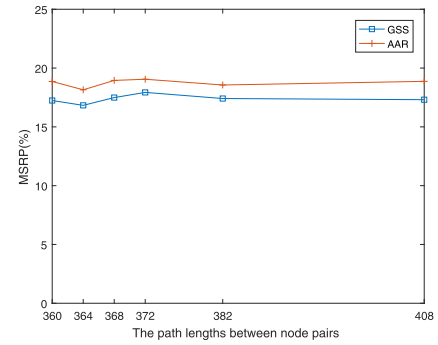


(d) MSRP with the same idle time slot size

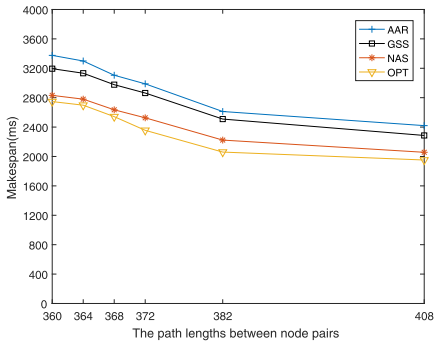
Fig. 8. The makespan and MSRP under different path lengths between node pairs with the sizes of data randomly generated in 1-10MB.



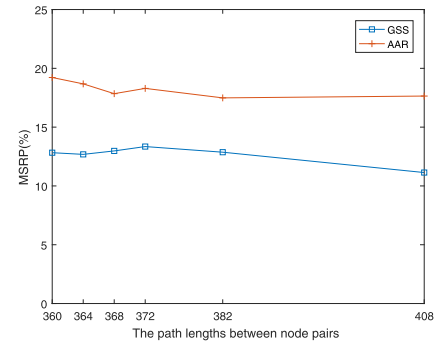
(a) Makespan with different idle time slot sizes



(b) MSRP with different idle time slot sizes



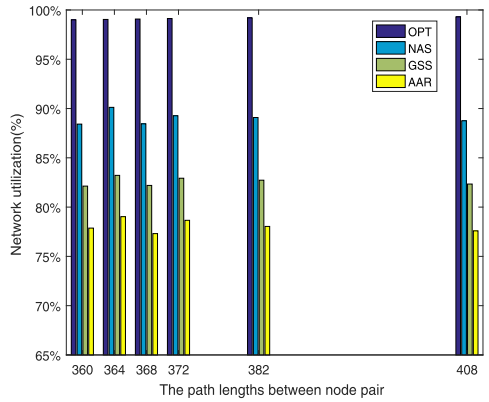
(c) Makespan with the same idle time slot size



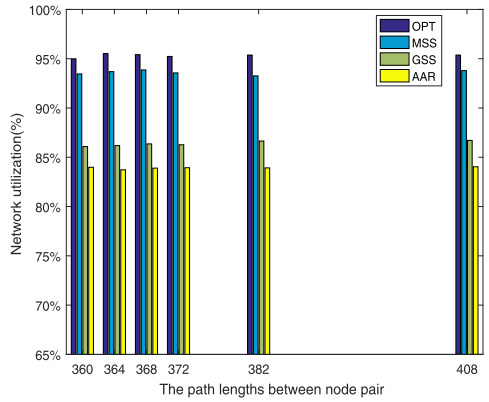
(d) MSRP with the same idle time slot size

Fig. 9. The makespan and MSRP under different path lengths between node pairs with the sizes of data randomly generated in 1-100MB.

increase of the number of node pairs. Fig. 7(a) depicts that the network utilization of algorithms OPT, NAS, GSS, and AAR with the different idle time slot sizes keeps at about 98%, 88%, 81%, and 77%, respectively. Similarly, Fig. 7(b) illustrates the network utilization of the four algorithms is approximately 96%, 93%, 86%, and 84% when the durations of idle time slots



(a) The performance with different idle time slot sizes



(b) The performance with the same idle time slot size

Fig. 10. The network utilization under different path lengths between node pairs.

are the same. Algorithm NAS performs better than algorithms ARR and GSS, and algorithm AAR obtains the lowest network utilization.

3) Impact of Various Path Lengths Between Node Pairs: We study the impact of various path lengths between node pairs on the performance of different algorithms. We have 12 mappers and 6 reducers, which means 72 node pairs, with 25 data to be transferred between each node pair. The paths are generated by controlling the sum of variable lengths between the node pairs. For example, the 72 node pairs consist of 64 pairs with path length 6, 4 node pairs with path length 4, and 4 node pairs with path length 2. Therefore, the sum of path lengths between node pairs is $64 \times 6 + 4 \times 4 + 4 \times 2 = 408$. To conduct the performance comparison under different data sizes, the sizes of data are randomly generated in 1-$10MB$ and 1-$100MB$ as shown in Fig. 8 and Fig. 9, respectively.

Figs. 8(a), 8(c), 9(a) and 9(c) show the makespan performance by varying the path lengths between node pairs. In general, the data transmission completion time decreases, as the average path length increases. Algorithm NAS achieves better performance than algorithms AAR and GSS, since NAS schedules the shuffle data by jointly considering path conflicts and future network states. Figs. 8(b) and 9(b) demonstrate that algorithm NAS outperforms both algorithm AAR and GSS by about 18% with different idle time slot sizes, while Figs. 8(d) and 9(d) show that the performance of algorithm NAS is about 18% and 12% better than that of algorithms AAR and GSS, respectively, with the same idle time slot size. Algorithm NAS obtains worse results than OPT by 14% and 18% with different idle time slot sizes, when the data sizes are randomly generated in 1-$10MB$ and 1-$100MB$, respectively. With the the same idle time slot size, algorithm NAS achieves close performance to OPT.

The network utilization of the four algorithms is evaluated with the data randomly generated in the size range of 1-$100MB$. There are 12 mapper nodes and 6 reducer nodes, and 25 data are to be transferred between each node pair. Fig. 10 shows that the network utilization of the four algorithms keeps steady with the increasing path lengths between the node pairs. Algorithm NAS can always make better use of network resources than algorithms AAR and GSS. When the durations of idle time slot are different, the network utilization of OPT, NAS, GSS, and AAR is about 99%, 88%, 81%, and 77%, respectively, as shown in Fig. 10(a). With the same idle time slot size, OPT, NAS, GSS, and AAR obtain the network utilization of about 96%, 94%, 86%, and 84%, respectively, and OPT and NAS achieve the similar performance.

VI. Conclusions

During the shuffle phase of MapReduce jobs, a significant amount of communication is required between the nodes running map and reduce tasks. The shuffle time accounts for a big part of the total running time of MapReduce jobs. Therefore, optimizing the makespan of shuffle phase can greatly improve the performance of MapReduce jobs. A large fraction of production jobs in data centers are recurring with predictable characteristics, and the recurring jobs split the network into periodic busy and idle time slots, which allows us to better schedule the shuffle data to reduce the makespan of shuffle phase by leveraging the future predictable periodic network status. In this paper, we formulated the shuffle scheduling problem with the aim to minimize the makespan of MapReduce shuffle phase, which is an $\mathcal{NP}$-complete problem. We proposed a simple and effective network-aware shuffle scheduling algorithm (NAS) to reduce the number of idle time slots required to transfer the shuffle data in order to reduce the shuffle makespan. We also proved that the proposed algorithm is a $\frac{3}{2}$-approximation algorithm to the shuffle scheduling problem when all the future idle time slots have the same duration. We finally conducted experiments through simulations, and the experimental results demonstrated the proposed algorithm could effectively reduce the makespan of MapReduce shuffle phase and increase network utilization.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] L. D. D. Babu and P. V. Krishna, "An execution environment oriented approach for scheduling dependent tasks of cloud computing workflows," *Int. J. Cloud Comput.*, vol. 3, no. 2, pp. 209–224, 2014

[3] Y. Guo, J. Rao, D. Cheng, and X. Zhou, "IShuffle: Improving Hadoop performance with shuffle-on-write," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1649–1662, Jun. 2017.

[4] S. Gault and C. Perez, "Dynamic scheduling of MapReduce shuffle under bandwidth constraints," in *Euro-Par 2014: Parallel Processing Workshops* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, Aug. 2014, pp. 117–128.

[5] A. Wadhonkar and D. Theng, "A survey on different scheduling algorithms in cloud computing," in *Proc. 2nd Int. Conf. Adv. Electr., Electron., Inf., Commun. Bio-Inform. (AEEICB)*, Feb. 2016, pp. 665–669.

[6] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 772–783, Apr. 2018.

[7] Y. Li and A. M. K. Cheng, "Transparent real-time task scheduling on temporal resource partitions," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1646–1655, May 2016.

[8] Z. Li, Y. Shen, B. Yao, and M. Guo, "OFScheduler: A dynamic network optimizer for MapReduce in heterogeneous cluster," *Int. J. Parallel Program.*, vol. 43, no. 3, pp. 472–488, Jun. 2015.

[9] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, and Y. Liu, "Multi-attributes-based coflow scheduling without prior knowledge," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1962–1975, Aug. 2018.

[10] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 407–420, Sep. 2015.

[11] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: Guaranteed job latency in data parallel clusters," in *Proc. 7th ACM Eur. Conf. Comput. Syst. (EuroSys)*, Apr. 2012, pp. 99–112.

[12] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[13] C. Hu, B. Liu, C. Xing, Z. Yue, L. Song, and M. Chen, "Queueing model based analysis on flow scheduling in information-agnostic datacenter networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[14] X. Zhang, M. Ding, and R. Wan, "PFO: Priority-based flow scheduling for online social network datacenter," in *Proc. IEEE 11th Conf. Ind. Electron. Appl. (ICIEA)*, Jun. 2016, pp. 500–505.

[15] Q. Zhou, P. Li, K. Wang, D. Zeng, S. Guo, and M. Guo, "Swallow: Joint online scheduling and coflow compression in datacenter networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 505–514.

[16] Y. Xu, H. Luo, and F. Ren, "Is minimizing flow completion time the optimal way in meeting flow's deadline in datacenter networks," *China Commun.*, vol. 13, no. Supplement 1, pp. 6–15, 2016.

[17] M. Khabbaz, K. Shaban, and C. Assi, "Delay-aware flow scheduling in low latency enterprise datacenter networks: Modeling and performance analysis," *IEEE Trans. Commun.*, vol. 65, no. 5, pp. 2078–2090, May 2017.

[18] B. Peng, J. Yao, Z. Qi, and H. Guan, "HybridPass: Hybrid scheduling for mixed flows in datacenter networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 1000–1009.

[19] W. Wang, Y. Sun, K. Salamatian, and Z. Li, "Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 1, pp. 5–18, Mar. 2016.

[20] P. Xiong, X. He, H. Hacigumus, and P. Shenoy, "Cormorant: Running analytic queries on MapReduce with collaborative software-defined networking," in *Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb)*, Nov. 2015, pp. 54–59.

[21] C.-Y. Lin and J.-Y. Liao, "An SDN app for Hadoop clusters," in *Proc. IEEE 7th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Nov./Dec. 2015, pp. 458–461.

[22] T. Liu, Y. Liu, P. Song, and D. Qian, "DScheduler: Dynamic network scheduling method for MapReduce in distributed controllers," in *Proc. IEEE 22nd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2016, pp. 208–215.

[23] J. Duan, Z. Wang, and C. Wu, "Responsive multipath TCP in SDN-based datacenters," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5296–5301.

[24] L.-W. Cheng and S.-Y. Wang, "Application-aware SDN routing for big data networking," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[25] C. X. Cai, S. Saeed, I. Gupta, R. H. Campbell, and F. Le, "Phurti: Application and network-aware flow scheduling for multi-tenant MapReduce clusters," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Apr. 2016, pp. 161–170.

[26] L. Yang, X. Liu, J. Cao, and Z. Wang, "Joint scheduling of tasks and network flows in big data clusters," *IEEE Access*, vol. 6, pp. 66600–66611, 2018.

[27] *CPLEX Optimizer*. Accessed: Feb. 28, 2020. [Online]. Available: http://https://www.ibm.com/analytics/cplex-optimizer

[28] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[29] Apache. *MapReduce Tutorial*. Accessed: Feb. 28, 2020. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html