IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 10, 2018, accepted October 26, 2018, date of publication October 31, 2018, date of current version November 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2878864

Joint Scheduling of Tasks and Network Flows in Big Data Clusters

LEI YANG[1], XUXUN LIU[2], JIANNONG CAO[3], (Fellow, IEEE), AND ZHENYU WANG[1]

[1]School of Software Engineering, South China University of Technology, Guangzhou 510006, China
[2]School of Electronic and Information Engineering, South China University of Technology, Guangzhou 510006, China
[3]Department of Computing, The Hong Kong Polytechnic University, Hong Kong

Corresponding author: Xuxun Liu (liuxuxun@scut.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61502312, in part by Hong Kong RGC under Grant 152244/15E, in part by the Science and Technology Fund of Guangzhou, China, under Grant 201802010025, in part by the Guangzhou Education Bureau-University Innovation and Entrepreneurship Platform Construction Key Project under Grant 2019PT103, and in part by the Fundamental Research Funds for the Central Universities.

ABSTRACT As an increasing number of big data processing platforms like Hadoop, Spark, and Storm appear and normally share the resources in the data center, it has been important and challenging to schedule various jobs from these platforms onto the underlying data center resources such that the overall job completion time is minimized. To solve the problem, the existing work either focus on the task-level scheduling techniques, such as Quincy and delay scheduling, or focus on the network flow scheduling techniques, such as D3 and preemptive distributed quick. These works deal with the scheduling of tasks and network flows separately and cannot achieve optimal performance. The reason is that the task scheduling without regard of the available network bandwidths may generate the task placement that causes serious network congestions and thus leads to long data transmission time. In this paper, we propose the joint scheduling technique by coordinating the task placement and the scheduling of network flows arising from these tasks. We develop a software-defined network (SDN)-based online scheduling framework which selects the task placement based on the available bandwidth on the SDN switches and at meanwhile optimally allocates the bandwidth to each data flow. Comprehensive trace-driven simulations show that the joint scheduling technique can take full use of the network bandwidth and thus reduce the job completion time by 55% on average compared with the benchmark methods.

INDEX TERMS Task scheduling, flow scheduling, data centers, software defined networks.

I. INTRODUCTION

Big data compute clusters running thousands of servers in the data center have become increasingly common over the past years. Many parallel data processing frameworks from the industry and academic organizations such as Hadoop [20], Dryad [21], Spark [22], Storm [23] and so on appear and are deployed on the clusters. The platforms process massive data and runs tens of thousands of jobs every day. Since the jobs share and compete for the data center resources, it is important to effectively schedule the jobs from the platforms to the underlying data center resources such that the job completion time is minimized.

Many job scheduling techniques have been proposed to solve the problem. Quincy [7] and Delay Scheduling [5] try to schedule the tasks close to the input data, aiming to increase the data locality while guaranteeing the basic fairness between the jobs. Due to the unbalanced distribution of input data across the server, data locality means that the tasks could be executed with unbalanced workloads on the servers which would result in long waiting time of tasks and low server utilization. To address the issue, Venkataraman *et al.* [8] develop the data-aware scheduling technique. The technique schedules the task to the server which minimizes the time it takes to transfer the input data. It reduces the job completion time through achieving optimal trade-off between the data locality and load balancing among the servers.

Although existing job scheduling techniques [3], [5], [7]–[10], [16] make sustainable efforts to improve the job completion time, they still have limitation. Since the job are often composed of a large number of tasks which have data dependence between each other, the job completion time is

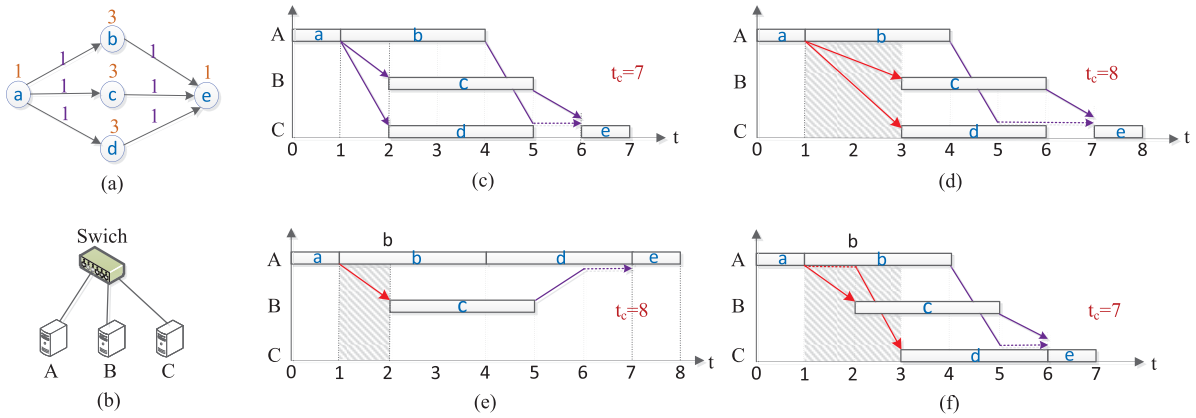


FIGURE 1. Example to motivate the joint scheduling of tasks and data transmission.

decided by the task execution time on the servers and as well the data transmission time over the data center networks. The job scheduling technique should guarantee that the servers can quickly process the tasks and at meanwhile the data flows arising from these tasks are transferred in the network as fast as possible [2], [4], [26]. However, the existing techniques mainly focus on the task level scheduling without regard of the dynamic network bandwidth, so they may generate the task placement that causes serious network congestions and thus leads to long data transmission time.

In this paper, we study the joint scheduling of the tasks and the network flows arising from these tasks. We design a Software Defined Network (SDN) based framework that coordinates the placement of tasks and the scheduling of flows by using the centralized SDN controller, and the online scheduling algorithm. The algorithm is triggered by the task completion events and flow completion events. It acquires the available bandwidths on the SDN switches and then assign the task to the server that can execute it at the earliest time. At meanwhile the algorithm adopts two important principles, i.e., to transmit As Early As Possible (AEAP) and As Fast As Possible (AFAP), to take full use of the bandwidth resources to speed up the job execution.

We conduct comprehensive simulations based on the realistic Facebook workloads over two weeks in 2009 to evaluate the online scheduling solution. We compare the joint scheduling technique with a set of 4 benchmark methods which separately schedule the tasks and network flows at different layers. We select the Delay Scheduling [5] and Heterogeneous Earliest Finish Time (HEFT) [6] as task scheduling algorithms in the benchmarks, and TCP and Preemptive Distributed Quick (PDQ) [14] as network flow scheduling algorithms. The results show that the joint techniques have near 2X better performance than the benchmark methods in terms of the average job completion time. We summarize the contributions in this paper as follows.

- To the best of our knowledge, this work is the first one to model and formulate the joint scheduling of tasks and network flows in data centers. We show that the problem is novel and distinguished from the previous job scheduling problems that focus on the task-level scheduling without concerning on the network resources and congestions.
- We develop a SDN based framework to coordinate the tasks scheduling and network bandwidth allocation, and further design the corresponding online scheduling algorithm which is triggered by the task completion and flow completion messages.
- We evaluate the online scheduling algorithm using realistic workload traces at Facebook, and compare it with four benchmark methods through extensive simulations. The results show the algorithm significantly outperforms the benchmark methods in the average job completion time.

II. MOTIVATION EXAMPLE

Fig.1 shows a simple example to motivate why we need to jointly consider the scheduling of data transmission in task scheduling. Suppose we schedule a job onto three identical machines. The three machines are connected through one switch, which is shown in Fig.1(b). Fig.1(a) shows the task graph for the job, where the node represents the task and the edge represents the data dependence between the tasks. The label of each node denotes the execution time of the task on the machines. The label on the edge denotes the data transmission time in the networks if the two connective tasks of the edge are not executed on the same machine. We assume the data transmission time labeled in the graph is calculated without network congestions. Now we want to schedule the tasks onto the machines such that the completion time of the job is minimized. Assume that the data transmission time of each edge is static, which means it does not change depending on the traffic in the networks, then we can easily obtain the optimal task scheduling shown in Fig.1(c). The completion time under the optimal scheduling is $t_c = 7$.

From the scheduling policy in Fig.1(c), we found the data flow $a \rightarrow c$ and data flow $a \rightarrow d$ occurs concurrently in the networks. Both data flows go through the same switch and compete for the bandwidth resources on the switch. In this situation, we call network congestion happens.

TABLE 1. Mathematical notations in this paper.

i	index of the job;
N	the number of jobs;
τ	index of the time slot (cloud and cloudlets);
T	the whole time period we consider;
$\mathcal{G}_i = (\mathcal{E}_i, \mathcal{V}_i)$	the task graph of the job i; $\mathcal{E}_i$ is the set of edges, and $\mathcal{V}_i$ is the set of tasks;
$C_{i,j}$	the execution time of the j-th task of the job i;
$D_{i,u,v}$	the amount of data on the edge (u, v) in the task graph of job i;
M	the number of servers in the data center;
$\mathcal{S}$	the set of switches in the data center network;
R_s	the bandwidth capacity of the switch s;
$t_{i,j}$	the time that the j-th task of the job i starts to execute;
$m_{i,j}$	the server where the j-th task of the job i is executed;
$f_{i,u,v}$	the corresponding data flow of the edge (u, v) in the task graph of the job i, if the tasks u and v are scheduled to different servers;
$\mathcal{S}_f$	the set of switches on the path of the flow f;
$\mathbb{R}_f(\tau)$	the transmission rate of the flow f at the time τ;

The transmission time of both flows would not achieve the ideal value labeled in the task graph. Assuming the fair sharing policy for the bandwidth, which has been used in most existing networks, the practical transmission time of both flows are doubled. The completion time under the 'optimal' scheduling in practical networks is increased to $t_c = 8$, which is shown in Fig.1(d). Fig.1(f) shows a better solution through the scheduling of data transmissions $a \rightarrow c$ and $a \rightarrow d$. We begin the data transmission $a \rightarrow d$ until the data transmission $a \rightarrow c$ is finished. The start time of task 'c' is one time unit earlier, and the job completion time is reduced to $t_c = 7$.

Can we develop the task scheduling to avoid the congestion, while achieving the original minimum completion time $t_c = 7$? Unfortunately, this may not always be realized. Fig.1(e) shows the task scheduling to avoid the congestion, where $t_c = 8$. In this example, we found that the minimum completion time of all the task scheduling with no congestion is $t_c = 8$. Therefore, only by scheduling the tasks in networks could not achieve the minimum completion time. We need to jointly consider the scheduling of underlying data transmissions.

III. SYSTEM MODEL AND PROBLEM STATEMENT

At first, we model the jobs to be scheduled. We consider a slotted time period, which is divided into T slots. Each time slot is indexed by τ, where $0 \leq \tau < T$. At each time slot, several jobs arrive at the DCN. Assume the total number of jobs arriving at the system is N during the whole time period. We use i to index the jobs, where $0 \leq i < N$. Each job is composed of a set of dependent tasks. Normally we use a Directed Acyclic Graph (DAG) $\mathcal{G}_i$ to represent the job i, in which the node represents the task, and the directed edge indicates the precedence among the tasks. Suppose the number of tasks of job i is n_i. We have $\mathcal{G}_i = \{\mathcal{V}_i, \mathcal{E}_i\}$, where $\mathcal{V}_i = \{j | 0 \leq j < n_i - 1\}$ denotes the set of tasks of job i, and $\mathcal{E}_i = \{(u, v) | u, v \in \mathcal{V}_i\}$ denotes the set of edges. Let $C_{i,j}$ denote the workload of j-th task in job i, which is measured by time units/slots. $D_{i,u,v}$ denotes the weight of edge (u, v) in job i, which represents the data transmission amount between the two connective tasks.

Next we describe the structures and resources of Data Center Networks (DCNs). We consider a multi-rooted tree topology (e.g., Fig.2), which has been widely adopted in today's DCNs. In the topology, the servers are interconnected through three layers of switches, i.e., edge switches, aggregation switches and core switches, to overcome limitations in port densities from commercial switches. Meanwhile, the topology leverages a large number of parallel paths between any pair of edge switches to achieve good throughput for data flows, and as well to avoid single-point-failure of switches. In the DCNs, the tasks from all the jobs are allocated to the servers. If two dependent tasks from the same job are allocated to different servers, data transmission will occur between the servers and pass by the switches in the network, which is also named as a *flow* in our paper. Assume that the tasks are executed on the servers without allowing preemption. If one server is allocated to one task, the server will be occupied by the task until it is finished. Each server has the same processing capability. Thus, the execution time of particular task on any server will be the constant $C_{i,j}$.

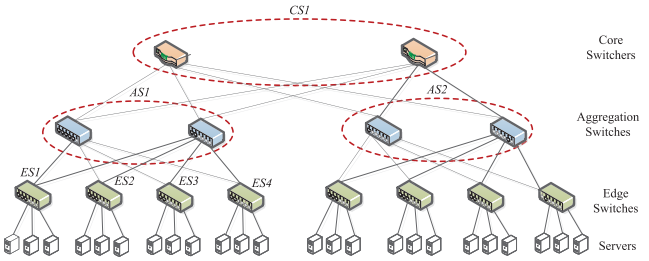


FIGURE 2. A common structure of data center networks.

The problem is to determine when and where (on which server) each task should be executed, and as well as schedule the flow of data transmission on the network links, such that the average completion time of jobs are minimized. Let $t_{i,j}$ denotes the time that the j-th task in job i starts to execute, and $m_{i,j}$ denotes the server where the j-th task in job i is executed, where $0 \leq t_{i,j} < T$ and $0 \leq m_{i,j} < M$. The completion time of j-th task in job i is $t_{i,j} + C_{i,j}$. We check every edge (u, v) in the DAG of each job i, if the two connective tasks u and v are not assigned to the same server, i.e., $m_{i,u} \neq m_{i,v}$, then a flow $f_{i,u,v}$ is to be scheduled in the network. Several properties of the flow are defined as follows.

- **Source**. It is defined as the *server* from which the flow starts. The source of flow $f_{i,u,v}$ is $m_{i,u}$.
- **Sink**. It is defined as the destination *server* where the flow arrives. The sink of flow $f_{i,u,v}$ is $m_{i,v}$.
- **Path**. It is defined as a sequence of *switches* the flow passes by. In the tree-based DNC structure, we use a set $\mathcal{S}_{f_{i,u,v}}$ of switches to represent the path of flow $f_{i,u,v}$. In the following descriptions, we sometimes neglect the subscripts, and denote it by $\mathcal{S}_f$.

- **Release time**. The release time of flow $f_{i,u,v}$ is defined as time that the data is ready by the precedent task u, which is represented by $t_{i,u} + C_{i,u}$.
- **Deadline**. It is defined as the latest time that the data transmission should be completed. The deadline of flow $f_{i,u,v}$ is represented by $t_{i,v}$.
- **Data amount**. The data amount of flow $f_{i,u,v}$ is $D_{i,u,v}$.
- **Rate**. The rate of flow is defined as the transmission speed, which is measured by data amount transmitted per time slot. The rate usually represents the network resources/bandwidth allocated to the flow. In our model, the rate of flow can vary with time. Let $\mathbb{R}_{f_{i,u,v}}(\tau)$ (or $\mathbb{R}_f(\tau)$ in short) denote the rate of flow $f_{i,u,v}$ at time slot τ. Note that the rate is a non-negative value. $\mathbb{R}_f(\tau) = 0$ means the flow suspends at time τ. If we have $\sum_{\tau=t_{i,u}+C_{i,u}}^{t_{i,v}} \mathbb{R}_f(\tau) = D_{i,u,v}$, we say that the flow can meet the deadline under the rate allocation $\mathbb{R}_f(\tau)$. Suppose $\mathbb{R}_f(\tau) = 0$ when $\tau < t_{i,u} + C_{i,u}$ or $\tau > t_{i,v}$.

Given the task scheduling decisions $t_{i,j}$ and $m_{i,j}$, can we find feasible flow schedule to meet their deadline requirement? If we can, we say $t_{i,j}$ and $m_{i,j}$ are feasible solution to our problem. Whether the flows are schedulable with deadline guaranteed or not depends on the throughput constraint of the switches in Fig.2. Assume each switcher has a maximum throughput. The rate of flows that pass through the switcher is then constrained by the switch's maximum throughput. In practical muti-rooted tree topology, each flow have multiple candidate paths. The flow scheduling is actually to select the path, and assign the transmission rate to the flow. There exist many protocols to solve the path selection problem [25]. In our problem, we neglect the path selection by simplifying the multi-rooted tree topology into a single rooted tree topology. In specific, at the core switches layer, we consider all the switches as one virtual switch whose maximum throughput is the summation of individual switches. We do the same simplification at the aggregation switcher layer. Since the simplification does not change the throughput constraint from the switches, we can easily approve that if the flows can be scheduled in the single-rooted tree topology, the flows will be schedulable in the multi-rooted tree topology.

After the topology simplification, each flow between two servers have single fixed path. We still use $\mathcal{S}_f$ to denote the dedicated path of flow f in the simplified tree topology. Let $\mathcal{S}$ denote the set of switches, which includes all the non-leaf nodes in Fig.2. Each individual switch in $\mathcal{S}$ is denoted as s. The maximum throughput of switch s is denoted by R_s. We formulate the scheduling problem as follows,

$$\min_{t_{i,j}, m_{i,j}, \mathbb{R}_f(\tau)} \sum_{i=0}^{N-1} \max_{j \in \mathcal{V}_i} \{t_{i,j} + C_{i,j}\} \tag{1}$$

$$\text{s.t. } \forall i \in [0, N-1], \quad \forall (u,v) \in \mathcal{E}_i, \; t_{i,u} + C_{i,u} \le t_{i,v} \tag{2}$$

$$\forall i, i' \in [0, N-1], \quad \forall j \in \mathcal{V}_i, \; \forall j' \in \mathcal{V}_{i'}$$
$$|m_{i,j} - m_{i',j'}| \times IN + (t_{i,j} - t_{i',j'} - C_{i',j'}) \times (t_{i,j} + C_{i,j} - t_{i',j'}) \ge 0, \tag{3}$$

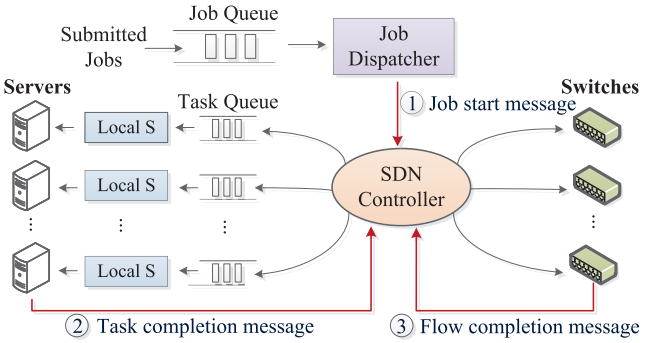


FIGURE 3. SDN based system architecture.

$$\forall i, \quad \forall (u,v) \in \mathcal{E}_i,$$
$$(m_{i,u} - m_{i,v})(\sum_{\tau=t_{i,u}+C_{i,u}}^{t_{i,v}} \mathbb{R}_f(\tau) - D_{i,u,v}) = 0 \tag{4}$$
$$\forall \tau \in [0, T-1], \forall s \in \mathcal{S}$$
$$\sum_{f_{i,u,v}} [\mathbb{R}_f(\tau) \times \mathbb{P}(s, S_f)] \le R_s \tag{5}$$

where IN in Equation (3) is a great positive constant which approaches to infinity, and $\mathbb{P}$ is a function of s and S_f. If $s \in S_f$, $\mathbb{P}(s, S_f) = 1$; otherwise $\mathbb{P}(s, S_f) = 0$.

Note that Equation (2) indicates that the tasks' execution time should satisfy the precedence constraint in the jobs. Constraint (3) shows that the server executes only one task at one time. If multiple tasks are scheduled to the same server, the tasks will be executed sequentially. Equation (4) represents the rate allocation of flows should guarantee the transmission of required amount of data before the deadlines. Equation (5) reflects each switch's maximum throughput can not be exceeded by the flows.

IV. SDN BASED FRAMEWORK AND ONLINE SOLUTION

In this section, we develop the online solution to the joint scheduling problem of tasks and data flows. We first give an overview of the system architecture, and then present the details of the solution.

A. SYSTEM ARCHITECTURE

Fig.3 shows the system architecture. It consists of three components: job dispatcher, SDN controller, and local schedulers on the servers. The newly submitted jobs by users are first added into the job queue. The job dispatcher periodically fetches the job from the queue, and triggers the SDN controller to schedule the tasks of the selected job. The SDN controller runs core scheduling algorithms, and takes charge of assigning the tasks to the servers, and meanwhile allocating the bandwidth resources of switches to the associated flows. The local scheduler on the server simply gets the active task in the queue to execute. Active means the dependent data of the task have been transmitted to the server; otherwise the tasks whose data have not arrived at the server is inactive and blocked in the task queue.

The SDN controller is triggered by three types of messages. The first is the job start message from the job dispatcher. After

triggered by the job dispatcher, the SDN first assigns the start tasks of the job to the idle server or the server with the least workload. Once the task is finished by the server, the SDN controller receives a task completion message from the sever as a trigger signal and schedules the successors of the task. The scheduling considers both the available bandwidth on the switches and the workload of the servers' task queues, and always assigns the task to the server where it can be executed at the earliest time. At the same time, the associated flow is assigned with as much bandwidth as possible that the switches can provide.

Besides being triggered by the job dispatcher and the task completion messages from servers, the SDN controller is also triggered by the flow completion message. Once the flow is finished, the released bandwidth resources of the switches are immediately re-allocated to other flows in the network. The core idea of our method is to let the data transmit As Early As Possible (AEAP), and As Fast As Possible (AFAP). By following the two principles, we can fully utilize the bandwidth resources to reduce the job completion time. We describe the two principles with details in the subsequent parts.

B. JOB DISPATCHER

All the newly submitted jobs are inserted into the job queue. The job dispatcher selects the job from the queue, and invokes the SDN controller to execute the selected job, i.e., to further allocate the tasks to the servers and the flows to the switches. We use the Highest Response Ratio Next (HRRN) policy to select the job from the job queue. The priority of each job is dependent on its estimated execution time, and also the amount of time it has spent waiting. The job execution time can be approximated by the summation of all the tasks' execution time in the job. Suppose the waiting time of job i is w_i. The priority of the job

$$p_i = \frac{w_i}{\sum_j C_{i,j}}. \tag{6}$$

The job with short execution time has high priority, and meanwhile the job gains high priority if it waits a long time. The job dispatcher controls the rate of delivering jobs to the data center. The rate depends on the number of jobs being executed, the loads of the servers, and the available bandwidth of the switches.

C. SCHEDULER ON SDN CONTROLLER

The SDN controller includes two scheduling algorithms which are respectively triggered by the task completion message and flow completion message. It relies on the following information to schedule the tasks and flows: *job execution list*, *flow list* and the available bandwidth of the switches.

The *job execution list* contains all the jobs which are being executed in the data center, which is shown in Fig.4(a). When the job dispatcher selects one job and delivers it to the SDN Controller, the SDN Controller adds the job into its *job execution list*. Once the job is finished, the SDN controller removes it out of the *job execution list*. For each job,

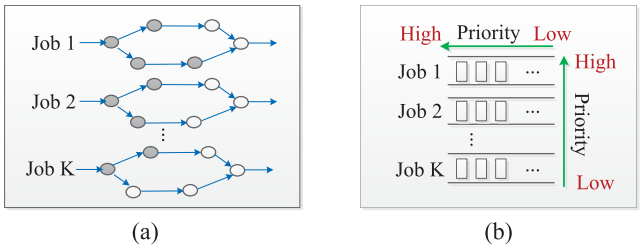


FIGURE 4. The two critical data structures maintained by SDN Controller. (a) Job execution list. (b) Flow list.

the SDN Controller maintains its task graph which indicates the precedence among the tasks, and the states of the tasks. There exist 6 types of states in total for each task: *unscheduled*, *pre-scheduled*, *scheduled*, *in execution*, and *completed*. *Unscheduled* represents the task is not scheduled by the SDN Controller. *Pre-scheduled* means that the task has been allocated by the SDN Controller to the server, but the allocation could be changed to other servers in future. *Scheduled* means the tasks have been allocated to servers, and the allocation can not be changed in future. *In execution* represents the task is being executed by the server. *Completed* means the task has been finished by the server. Fig.5 shows the state transition graph.

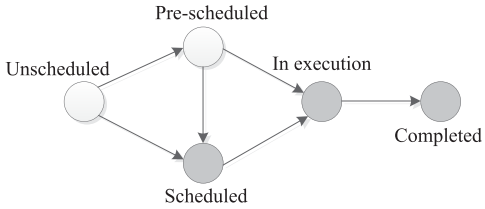


FIGURE 5. The state transition graph of the tasks.

The *flow list* includes all the data flows in the data center networks. Each flow is allocated with certain bandwidth when it is added into the *flow list*. Once one flow is finished, which means all the required data has been transmitted from the source server to the destination server, it is deleted from the *flow list*. The released bandwidth by the flow is immediately allocated to other flows. The flows have different priorities to obtain the released bandwidth.

We use a two-dimensional priority queue to abstract the flows' priorities as shown in Fig.4(b). If two flows are from different jobs, we then compare the start execution time of the jobs. The earlier the job starts to execute, the higher priority the flow from the job has. If the flows are from the same job, we assign the priorities based on the job's task graph. Assume in the task graph, the node is labeled with the execution time, and the edge is labeled with the data transmission time, which is normalized using the amount of data transmission divided by the average bandwidth capacities of all the switches. For each edge in the task graph, we find the longest path (including the edge itself) to the end node. The length of the path represents, if the edge/flow is scheduled to transmit

now, at least how much time are needed until the whole job is completed. The longer the path is, the higher priority the corresponding edge has.

1) TASK COMPLETION MESSAGE TRIGGERED SCHEDULING

Once one task is finished on the server, the SDN controller obtains the relevant job's task graph, and schedules the successors of the finished task onto the servers. We use the Earliest Start Time (EST) policy in the scheduling. In EST, among all the servers, we always assign the task to the server that can start the execution at the earliest time. After the allocation of the successor, the SDN controller inserts the corresponding data flow into the *flow list*, and assigns as much bandwidth as possible to the flow, according to the available bandwidth of the switches on the flow's path. The completed task's state is then updated as *scheduled* accordingly. Note that if the successor is scheduled on the local machine, the successor's state is updated as *pre-scheduled*. Meanwhile the corresponding flow is still added into the *flow list*, but it does not occupy any bandwidth resources. We name this kind of flows without bandwidth occupied as **virtual flows**. Correspondingly, the other flows are named as **real flows**. The case of local execution often happens when the network does not have much bandwidth at the scheduling time. The SDN controller tentatively assigns the task to the local machine. In future, if more bandwidth resources are released in the network and meanwhile the execution of task does not start, the task could be changed to the other servers. The *virtual flow* changes to be *real flow*.

2) TRANSMIT AS EARLY AS POSSIBLE (AEAP)

As long as the task is completed, our algorithm immediately determines the places where the successors are executed and begins the data transmission if the successors are allocated to different servers from the completed tasks. The purpose is to fully utilize the available network bandwidth and start the data transmission as early as possible. If the completed task's successor has single in-coming edge, e.g., as shown in Fig.6(b), it is easily to schedule the successor task 2 immediately after the task 1 is completed. However, if the completed task's successor has multiple in-coming edges, e.g., as shown in Fig.6(b), the case becomes complex. Task 2 could not be scheduled immediately after the task 1 is completed, because at this moment the other precedent tasks 3 and 4 of the task 2 may not have been scheduled. It is hard to determine the best server for the task 2 according to the EST.

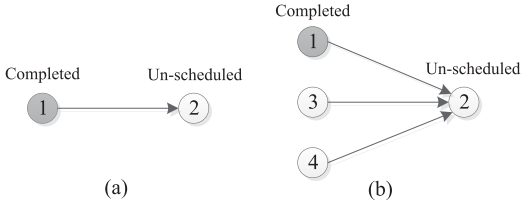


FIGURE 6. The illustration of AEAP principle.

Algorithm 1 The Task Completion Message Triggered Scheduling Algorithm

Input : the completed task p which triggers the scheduler, the task graph, the available bandwidth of the switches
Output: scheduling decisions of p's successors including their states and allocated servers

```
Update the state of p to be completed;
for each out-going edge e of the task p do
    if The state of e's tail task q is scheduled then
        if The task q has been scheduled on different server from the task p then
            Insert the data flow e into the flow list;
        continue ;
    if The task q has single in-coming edge then
        Decide the server m to execute q by Equation (7);
        if m is the server where the task p is executed then
            Update the state of q to be pre-scheduled;
            Insert the virtual flow e into the flow list;
        else
            Update the state of q to be scheduled;
            Insert the real flow e into the flow list;
        * Allocate q to the task queue of m;
    else
        S ← 0;
        for each in-coming edge e' of the task q do
            if The state of the head task of e' is unscheduled or pre-scheduled then
                S ← 1;
                break ;
        if S == 1 then
            continue ;
        Decide the server m to execute q by Equation (10);
        Update the state of q to be scheduled;
        * Allocate q to the task queue of m;
        for each in-coming edge e' of the task q do
            if the state of the head task of e' is completed, and it has been executed on the different server from task q then
                Insert the edge e' into the flow list;
return;
```

In existing online scheduling approaches, the task 2 is usually scheduled after all the precedent tasks e.g., task 1, 3, and 4, are completed. That means the data flows (1, 2), (3, 2) and (4, 2) do not begin to transmit in the network until all the tasks 1, 3, and 4 are finished. Obviously the existing methods

delay the data transmissions which could have been started earlier and thus reduce the task 2's completion time. In our algorithm, when the task 1 is completed, we check the state of the other precedent tasks of the task 2, e.g., task 3 and task 4. As long as all the tasks' state are 'scheduled', or 'in execution' or 'completed', we will schedule the task 2 immediately; otherwise if any of these tasks' states is 'unscheduled' or 'pre-scheduled', we will not schedule the task 2 at the time when the task 1 is finished. That is to say, the scheduling the task 2 does not wait for the completion of all the precedent tasks. Instead our algorithm schedules the task 2 as long as the precedent tasks' hosting servers are determined. The server is selected to execute the task 2 according to the EST policy which is explained with details in Section IV-C.4. The associated flows are then inserted to the *flow list*.

3) TRANSMIT AS FAST AS POSSIBLE (AFAP)

Another important design principle of our algorithm is that, when the flow is newly added into the *flow list*, we always allocate the maximum bandwidth that the switches can provide at current to the flow, such that it can transmit the data as fast as possible. We do not specially reserve bandwidths for the future. The purpose is to increase the bandwidth utilization of the networks. For each newly flow, the allocated bandwidth equals to the minimum available bandwidth among all the switches on the flow's path.

4) SERVER SELECTION–EARLIEST START TIME

In our method, we select the server for the task by using the Earliest Start Time (EST) policy. For example, In Fig.6(a), when task 1 is completed, we need to select one from all the servers to execute the task 2. The server which can start to execute the task 2 at the earliest time is selected. Let (i, u) denote the finished task of job i, and (i, v) denote task (i, u)'s successor in the task graph of job i. The execution server of the task (i, u) is $m_{i,u}$. The SDN Controller determines the server $m_{i,v}$ to execute the task (i, v) by the following equation

$$m_{i,v} = \arg\min_{m=1}^{N} \max\{t_{TRAN}(m_{i,u}, m), t_{AVL}(m)\}, \tag{7}$$

where $t_{TRAN}(m_{i,u}, m)$ denotes the data transmission time if the task (i, v) is allocated to the server m; $t_{AVL}(m)$ denotes the earliest available time of the server m. With the principle of transmitting As Fast As Possible(AFAP), the data transmission time $t_{TRAN}(m_{i,u}, m)$ is estimated according to the available bandwidth of each switches by Equation (8)

$$t_{TRAN}(m_{i,u}, m) = \frac{D_{i,u,v}}{\min_s B_s}, s \in Path[m_{i,u} \rightarrow m], \tag{8}$$

where B_s represents the free bandwidth that the switch s has.

If the task to be scheduled has multiple precedent tasks, e.g., the task 2 in Fig.6(b), we select the server for the task when all the precedent tasks have been scheduled. The server which can start the task at the earliest time is selected. For each candidate server, the start time depends on three factors: the completion time of the precedent tasks, the data transmission time from the precedent tasks to the server, and the available time of the server. Suppose $\mathcal{T}$ denotes the set of precedent tasks, e.g., $\mathcal{T} = \{1, 3, 4\}$ in Fig.6(b). For each precedent task $u \in \mathcal{T}$, the estimated completion time is c_u. The start time of the task on each server m can be formulated by

$$t(m) = \max\{\max_{u \in \mathcal{T}}\{c_u + t_{TRAN}(m_u, m)\}, t_{AVL}(m)\}. \tag{9}$$

We select the server with the earliest start time

$$m^* = \arg\min_{m=1}^{N} t(m). \tag{10}$$

5) FLOW COMPLETION MESSAGE TRIGGERED SCHEDULING

Once one flow is finished, the released bandwidth is immediately allocated to other flows. As the flows are assigned with various priorities, the flow with the highest priorities obtains as much bandwidth as possible at the earliest time, and then it turns to the flows with lower priorities until all the released bandwidth are allocated. Note that the *real flows* have been assigned with certain bandwidth, it seeks the best effort to increase the bandwidth when it gets turn to obtain the released the bandwidth. When it turns to the *virtual flow* to obtain the released bandwidth, if the head task of the flow/edge is not executed, the SDN controller decides the destination server of the flow again according to the EST policy; otherwise if the precedent task has been executed, the SDN controller removes the flow out of the *flow list*. Algorithm 2 shows the details of the flow completion message triggered scheduling.

D. LOCAL SCHEDULER ON SERVERS

The local scheduler maintains the *task queue* of the server. We define two states for the tasks in the task queue, i.e., *ready* and *blocked*. *Ready* indicates that the task can be executed by the server currently. *Blocked* means that the task can not be executed currently since the data from the precedent tasks do not arrive at the server. When the task is initially allocated by the SDN controller to the task queue, if the state of the task maintained by SDN controller is *pre-scheduled*, the task is then *ready* to execute in the task queue; otherwise if the state is *scheduled* on SDN controller, the task is then *blocked* in the task queue and waits for the data transmissions. The blocked task changes to be ready when all its dependent data arrives at the server. The local scheduler adopts the first-come-first-server policy, and always selects to execute the ready task which is added to the task queue at the earliest time.

V. EVALUATION

A. ENVIRONMENT SETUP

We evaluate the online solution based on the real world workloads. As follows we first describe environment settings including the network topology, the workloads and the simulator, and then present the results. In the evaluation environment, we use single-rooted tree data center topology which is shown in Fig.8. The topology contains 96 servers which

Algorithm 2 The Flow Completion Message Triggered Scheduling Algorithm

Input : the *flow list*, the completed flow f, the available bandwidth of the switches
Output: re-allocation of released bandwidth to the flows in *flow list*

```
Remove the completed flow f from the flow list;
Increase the available bandwidth of the switches on the flow f's path;
Visit the flow e with the highest priority in the flow list;
while one of the switches has available bandwidth do
    if the visiting flow e is the real flow then
        if all the switches on e's path has available bandwidth then
            Increase transmission rate of flow e to the maximum;
            Update the available bandwidth of switches;
    else if the virtual flow e's tail task is in execution or completed then
        Remove e from the flow list;
    else
        Determine the server m for e's tail task by Equation (7);
        if m is different from the server where e's head task is executed then
            Change the flow e to be real flow;
            Assign the maximum bandwidth to e;
            Update the available bandwidth of switches;
            * Allocate e's tail task to the task queue of m;
    Visit the next flow e in the flow list;
return;
```

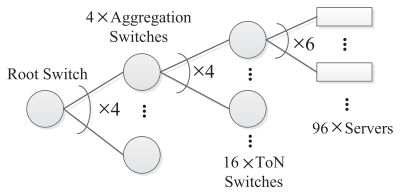


FIGURE 7. The network topology.

are organized into 16 racks with 6 servers per rack. Each rack has a top-of-rack (ToR) switch, and the ToR switches are connected through the aggregation switches with 4 ToR switches per aggregation switches. All the aggregation switches are connected by the root switch. The capacity of the root switch, aggregation switches, and the ToR switches are respectively 2 Gbps, 2 Gbps, and 3 Gbps.

FACEBOOK WORKLOADS

We use the realistic MapReduce workloads at Facebook over two weeks in 2009 [1]. We sampled the job inter-arrival times at random from the Facebook traces. The inter-arrival time follows the normal distribution with the mean of 15 seconds. The sampled workloads contain 200 jobs in total and have 50 minutes long. In the original data-set, each job includes the properties including the input data size, shuffle data size, output data size, the Map time and Reduce time. The Map/Reduce time is the total execution time of all the Map/Reduce tasks. To generate the detailed task graph for each MapReduce job, we assume that within one job the Map/Reduce tasks have the same execution time. For each job, we first randomly choose the execution time respectively for each Map task and Reduce task. Then, the number of Map tasks and Reduce tasks can be calculated accordingly. The data transmission size on each edge in the task graph can be computed based on the number of Map/Reduce tasks and the input/shulf/output data size. We note that the jobs in the workload trace are quite different in terms of the number of tasks. Table 2 lists the environment parameters in the evaluation.

TABLE 2. Parameters of the workloads.

Parameters	Values
The number of jobs	200
The average job size	1210
The mean of job submission period (*)	15 seconds
The average execution time of the task	13 seconds
The average data transmission size on the edges	196 MB
The number of servers (*)	96
The number of switches (*)	21
The average bandwidth capacity of switches (*)	2.8 Gbps

We compare our joint scheduling approach with the benchmark methods which deal with the task scheduling and flow scheduling independently. In particularly, we have developed our own event-driven simulator written in C++ to model the benchmark methods (Fig.8). The simulator consists of two layers: the task scheduling layer and the flow scheduling layer. At the task scheduling layers, we consider two existing algorithms: **delay scheduling** [5] and **HEFT** [6]. Delay scheduling is a well-known scheduling technique in big data clusters that guarantees the data locality by scheduling the tasks close to their input data. HEFT is a heuristic algorithm that particularly schedules the DAG structured jobs in the heterogeneous computing networks. It assigns the task to

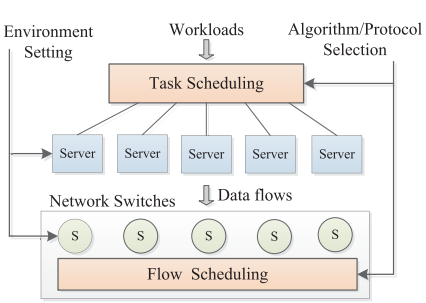


FIGURE 8. Diagram of the simulator for benchmark methods.

the server where it cloud be finished at the earliest time. At the network layer, we consider two data flow algorithms. One is the Fair Scheduling (**FS**) that allows the data flows in the network to share the bandwidth resources fairly. Most commonly used transportation control protocols in data centers like TCP, RCP and DCTCP [17] all emulates the fair scheduling. The other one is Shortest Job First (**SJF**) flow scheduling, which give priority to the short flow by pausing the contending flows. It is also used in existing transportation control protocol like PDQ [14]. The benchmark methods are the four different combinations of the selected task scheduling algorithms and flow scheduling protocols, which are denoted as follows: 1) **Delay Scheduling + FS**, 2) **HEFT + FS**, 3) **Delay Scheduling + SJF**, 4) **HEFT + SJF**.

B. RESULTS

The major performance metric is the **job completion time**. Fig.9shows the distribution of the job completion time of our method and the benchmark methods. Our proposed method outperforms the other benchmark algorithms significantly. In particular, the average job completion time of the joint scheduling is 48 seconds, while the average job completion time of the benchmark methods are larger than 106 seconds. We declare that our proposed method can decrease the job completion time by 55% compared with the benchmark methods. This is because our method jointly considers the network status, such as the flows in transmission and the available bandwidth resources, when doing the task scheduling decisions. However, the benchmark algorithms treat the task scheduling and flow scheduling independently. It first schedules the tasks onto the servers without considering the network congestions. Based on the task allocations, it relies the underlying network protocols to schedule the network flows. Among the benchmark methods, the 'HEFT + SJF' method has the best performance. The reason is that the HEFT relies on some prior knowledge of the job's task graph, i.e., the tasks' execution time and the amount of data transmission between the dependent tasks, during the online task scheduling, so it performs better than the delay scheduling. Meanwhile, SJF is demonstrated to have better performance than traditional FS in terms of the overall completion time.

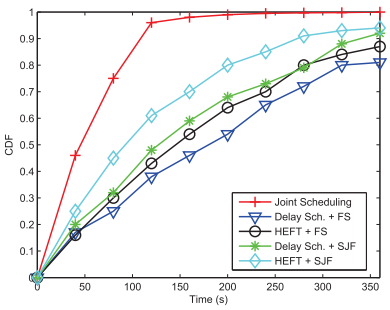


FIGURE 9. CDF of the job completion time.

We measure the resource utilization for the five methods. In particular, we respectively consider the utilization of two types of resources, i.e., servers and networks. The server utilization indicates how many percentages of all the server slots are busy; while the network utilization indicates the ratio of the occupied bandwidth on the switches and the total amount of bandwidth. Fig.10 shows that the joint scheduling has slightly higher server utilization than the benchmark methods. However, in terms of network resources, the joint scheduling has much higher utilization than the others. We conclude that the joint scheduling methods can fully take use of the network resources to improve the performance. The reason is that the benchmark methods do not take into account the network resources during the phase of task allocation. After the task scheduling, the data transmission workloads in the networks may fluctuate greatly over the time. The network sometimes is overloaded with a mass of flows, while at the other times the network deal with few data flows and many network resources are idle. The joint scheduling can avoid this problem, because it is able to adapt the task scheduling such that the data transmission workloads onto the network are more uniformly distributed over the time.

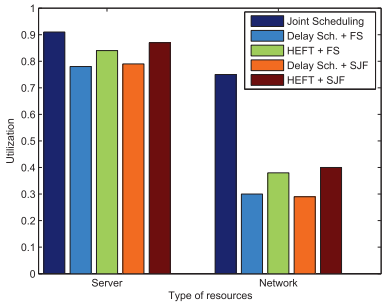


FIGURE 10. The server utilization and network utilization.

1) IMPACT OF THE SUBMISSION RATE OF THE WORKLOADS

In the evaluation setting, the 200 jobs are submitted with the interval of 15 seconds. To evaluate how the amount of workloads affect the performance, we change the job submission interval. Small interval means that the jobs are submitted to the data centers with high rate. Table 3 presents the performance results under various job submission rates. Fig.11a plots the performance of our proposed methods and the benchmarks methods. All the methods have decreasing average job completion time when the job submission rate decreases. It is interesting to find that the joint scheduling has high performance gain over the benchmark methods specially when the workloads are intensive. This is because in task scheduling when the jobs are submitted with high velocity, the network bandwidth competition among the data transmissions becomes as important as the competition for computing resources at the servers. The joint scheduling allocates the task with the regarding of network flow scheduling and congestion avoidance, and thus has better performance

TABLE 3. The performance results under different settings for the job submission rate.

Submission rate (Jobs/s)	$\frac{200}{9}$	$\frac{200}{12}$	$\frac{200}{15}$	$\frac{200}{18}$	$\frac{200}{21}$	$\frac{200}{24}$	$\frac{200}{27}$	$\frac{200}{30}$
Joint Scheduling	92s	66s	48s	44.3s	40.5s	36s	32s	30s
Delay Sch.+FS	210s	190s	172s	154s	137s	120s	104s	91s
HEFT+FS	191s	167s	152s	132s	114s	99s	85s	72s
Delay Sch.+SJF	185s	163s	145s	125s	108s	92s	78s	66s
HEFT+SJF	171s	137s	106s	94s	81s	69s	58s	48s

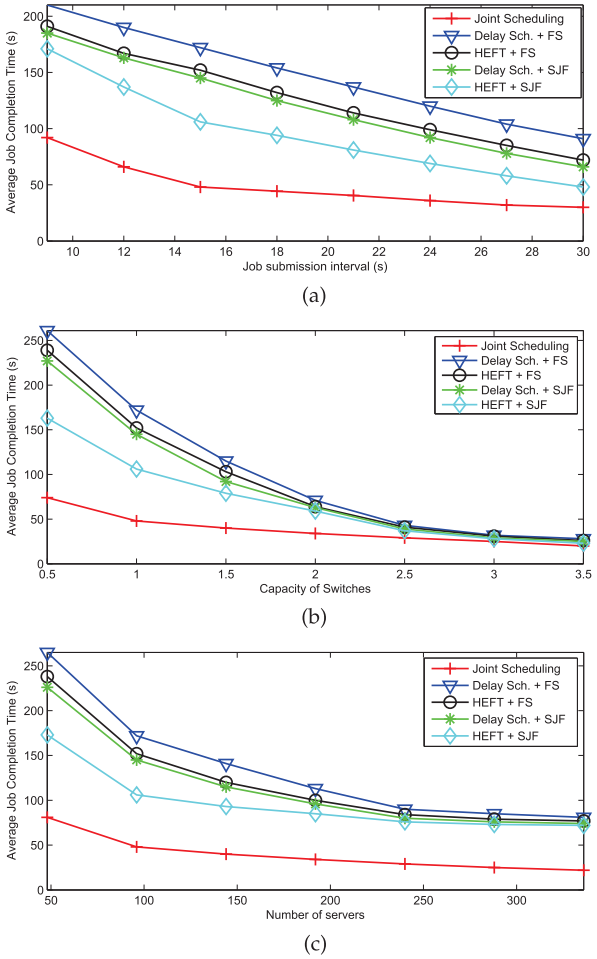


FIGURE 11. The joint scheduling approach outperforms the benchmark methods under various settings of job submission rate, network capacity, server number. (a) Submission rate of the workloads. (b) Network capacity. (c) The number of servers.

than benchmark methods that simply schedules the tasks by assuming static network conditions.

2) IMPACT OF THE NETWORK CAPACITY

Fig.11b shows the job completion time of the five scheduling methods under various network capacities. Note that the x-axis indicates the multiplies relative to the original capacity of switches, e.g., 1.5 means the capacity of each switch is 1.5 times of the original value. Corresponding numeric results are shown in Table 4. We find that our proposed method has much better performance over the benchmark methods when

TABLE 4. The performance under different settings for the network capacity.

	0.5X	1X	1.5X	2X	2.5X	3X	3.5X
Joint Scheduling	74s	48s	40s	34s	29s	25s	20s
Delay Sch.+FS	261s	172s	115s	71s	43s	32s	28s
HEFT+FS	239s	152s	103s	64s	41s	31s	26s
Delay Sch.+SJF	227s	145s	92s	63s	39s	29s	25s
HEFT+SJF	163s	106s	79s	59s	37s	28s	23s

the bandwidth resources are relatively scarce in data center. As the capacity of switches increase, the performance gain over the benchmark methods decrease. Specially when the network has enough resources, i.e., the capacity of switches is 3.5 time of the original setting, our methods almost have the same performance with the benchmark methods. This is because there exists no network congestions when the network has enough resources, and the performance is simply affected by the scheduling algorithm at the task scheduling layer. In practical data centers, the network resources always are scarce relative to the computation resources, in which case our proposed methods have obvious performance gain over the benchmark methods.

3) IMPACT OF THE NUMBER OF SERVERS

In the initial setting, the number of servers per rack is 6. Now we still keep the network topology as shown in Fig.7 and change the number of servers in each rack which varies from 3 to 31. The number of switches and the capacity of each switch do not change. The total number of servers varies from 48 to 336. Fig.11c shows how the performance changes depending on the number of servers. Table 5 presents the numeric performance results correspondingly. The joint scheduling approach significantly outperforms the benchmark methods under all the number of servers. It is shown that when the number of servers increase from 240 to 366, the benchmark methods nearly do not have performance gain. The reason is that when the number of servers are large enough, the bottleneck affecting the performance is the network resources. The benchmark methods that allocates

TABLE 5. The performance under different settings for number of servers.

	50	100	150	200	250	300	350
Joint Scheduling	81s	48s	40s	34s	29s	25s	22s
Delay Sch.+FS	265s	172s	141s	113s	90s	85s	28s
HEFT+FS	238s	152s	120s	100s	84s	79s	26s
Delay Sch.+SJF	226s	145s	115s	96s	80s	76s	25s
HEFT+SJF	173s	106s	93s	85s	76s	73s	23s

the tasks without considering the network congestions in the meanwhile do not enhance the performance simply by increasing the server number.

VI. RELATED WORK

In this section, we will describe the state-of-arts on the related topics including the task scheduling, network flow scheduling and SDN based scheduling techniques.

A. TASK SCHEDULING IN BIG DATA CLUSTERS

One of the related work is task scheduling in big data clusters. Many technqiues have been proposed to schedule the tasks for various purposes such as the fairness, data locality, throughput, scalability and so on. Quincy [7] and Delay scheduling [5] try to improve the data locality by scheduling the task close to the data, and meanwhile guarantee the fairness. Considering that the data could be distributed acrross multiple machines and it is not possible to co-locate the tasks with all its inputs, Venkataraman *et al.* [8] develop the data-aware scheduling techniques which aims to schedule the task at a machine that minimizes the time it takes to transfer the data.

Unlike these works that focus on the scheduling of tasks with the fixed data placement, Corral [9] coordinates the data placement and tasks of jobs for achieving better locality. There exists work that proposes to maximize the data processing throughput specially for the pipelined application workflows in big data clusters [12]. More recently a few techniques [10], [11] have been developed to solve the scalability problem in large-scale clusters, which are named as distributed scheduling techniques.

These work mentioned above focus on the task scheduling without regards of the underlying network protocols, while in our proposed technique the tasks are scheduled with considering the network workloads and congestions. We jointly schedule the scheduling of tasks and the flows such that the network bandwidths are fully utilized to reduce the job completion time. Chowdhury *et al.* [24] have the similar idea to the proposed joint scheduling, but they optimize the replica placement instead of the task scheduling based on the traffic on the network links during the writes to the cluster file systems.

B. NETWORK FLOW SCHEDULING

Another related work is network flow scheduling in data centers. TCP and DCTCP [17] are the commonly used data transfer protocols in data center. They emulate the fair sharing and can achieve satisfactory performance in terms of the network throughput. D3 [13] has been lately proposed to meet the deadlines of network flows which could not be guaranteed in previous TCP based protocols. PDQ [14] tries to emulate a shortest job first scheduling meachanism in a distributed way, in order to complete the flows quickly as well as meeting the deadlines. Varys [15] proposed the coflow abstraction and addresses the inter-coflow scheduling to reduce the communication time of data-intensive jobs. The recent work, Baraat [16], implements a distributed flow scheduling system that treats the flows generated from the same application together. However, the benefits from such network scheduling technique are inherently limited since the end-points of the network flows are fixed by applications. Our work jointly optimizes the task scheduling which determines the endpoints of flows and the flow scheduling. It has been demonstrated to obtain more performance gain in the data center applications.

C. SDN BASED TASK SCHEDULING TECHNIQUES

The most related work is the study of task scheduling in the SDN based big data clusters. Qin *et al.* [19] proposed the bandwidth-aware task scheduling technique by using the SDN. In this work, the tasks are allocated onto server at first and then the SDN takes charge of the bandwidth allocation lately. Like the previous method, the scheduling of tasks and network flows are still done independently. Alkaff *et al.* [18] proposed the cross-layer scheduling approach particularly for the cloud computing engines such as Storm and Hadoop. It leverages the centralized SDN controller to coordinate the placement of application tasks in tandem with the selection of the network routers arising from these tasks. However, it does not consider the allocation of bandwidth onto the flows, which we believe have much impact on the performance of the job scheduling.

VII. CONCLUSION

In this paper, we studied the joint scheduling problem to minimize the job completion time in data centers. The problem jointly optimized the scheduling of tasks and underlying network flows caused by these tasks. We have designed a SDN based framework and the online solution. The framework coordinates the task placement and the bandwidth allocations through a SDN controller. We have done extensive simulations based on the realistic workload traces at Facebook, and compares the joint scheduling with four benchmark methods which separately schedule tasks and network flows.The results show that our proposed method can decrease the job completion time by 55% on average compared with the benchmark methods. Furthermore, the joint scheduling has slightly higher server utilization than the benchmark methods. However, in terms of network resources, the joint scheduling has much higher utilization than the others. We conclude that the joint scheduling methods can fully take use of the network resources to improve the performance.

REFERENCES

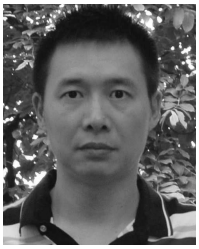
[1] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating mapreduce performance using workload suites," in *Proc. 19th IEEE Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst. (MASCOTS)*, Jul. 2011, pp. 390–399.

[2] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting massive D2D collaboration for energy-efficient mobile edge computing," *IEEE Wireless Commun.*, vol. 24, no. 4, pp. 64–71, Aug. 2017.

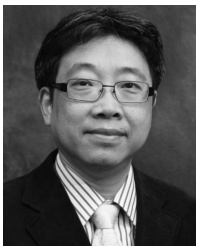
[3] K. Xie *et al.*, "Distributed multi-dimensional pricing for efficient application offloading in mobile cloud computing," *IEEE Trans. Service Comput.*, to be published, doi: 10.1109/TSC.2016.2642182.

[4] X. Liu, "A novel transmission range adjustment strategy for energy hole avoiding in wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 67, pp. 43–52, May 2016.
[5] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. EuroSys*, 2010, pp. 265–278.
[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[7] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. SOSP*, 2009, pp. 261–276.
[8] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica, "The power of choice in data-aware cluster scheduling," in *Proc. OSDI*, 2014, pp. 301–316.
[9] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proc. ACM SIGCOMM*, 2015, pp. 407–420.
[10] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proc. SOSP*, 2013, pp. 69–84.
[11] E. Boutin *et al.*, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proc. OSDI*, 2014, pp. 285–300.
[12] A. Benoit, Ü. V. Çatalyürek, Y. Robert, and E. Saule, "A survey of pipelined workflow scheduling: Models and algorithms," *ACM Comput. Surv.*, vol. 45, no. 4, Aug. 2013, Art. no. 50.
[13] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM*, 2011, pp. 50–61.
[14] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM*, 2012, pp. 127–138.
[15] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with Varys," in *Proc. ACM SIGCOMM*, 2014, pp. 443–454.
[16] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. ACM SIGCOMM*, 2014, pp. 431–442.
[17] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, 2010, pp. 63–74.
[18] H. Alkaff, I. Gupta, and L. M. Leslie, "Cross-layer scheduling in cloud systems," in *Proc. IEEE Int. Conf. Cloud Eng.*, Mar. 2015, pp. 236–245.
[19] P. Qin, B. Dai, B. Huang, and G. Xu, "Bandwidth-aware scheduling with SDN in Hadoop: A new trend for big data," *IEEE Syst. J.*, vol. 11, no. 4, pp. 2337–2344, Dec. 2017.
[20] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[21] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," *Operating Syst. Rev.*, vol. 41, no. 3, pp. 59–72, 2007.
[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, 2010, p. 10.
[23] A. Toshniwal *et al.*, "Storm@Twitter," in *Proc. Int. Conf. Manage. Data*, 2014, pp. 147–156.
[24] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *Proc. ACM SIGCOMM*, 2013, pp. 231–242.
[25] P. Wang, H. Xu, Z. Niu, D. Han, and Y. Xiong, "Expeditus: Congestion-aware load balancing in Clos data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 3175–3188, Oct. 2017.
[26] X. Liu and P. Zhang, "Data drainage: A novel load balancing strategy for wireless sensor networks," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 125–128, Jan. 2018.

LEI YANG received the B.Sc. degree from Wuhan University in 2007, the M.Sc. degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010, and the Ph.D. degree from the Department of Computing, The Hong Kong Polytechnic University, in 2014. He is currently an Associate Professor with the School of Software Engineering, South China University of Technology. His research interests include mobile cloud computing, edge computing, task scheduling, and resource management.



XUXUN LIU received the Ph.D. degree in communication and information system from Wuhan University, Wuhan, China, in 2007. From 2017 to 2018, he was a Visiting Scholar with the University of California, Berkeley, CA, USA. He is currently an Associate Professor with the School of Electronic and Information Engineering, South China University of Technology, Guangzhou, China. He has authored or co-authored over 30 papers in international journals and conference proceedings. His current research interests include wireless sensor networks, cyber physical systems, intelligent computing, and mobile computing.



JIANNONG CAO (M'93–SM'05–F'15) received the B.Sc. degree in computer science from Nanjing University, China, in 1982, and the M.Sc. and Ph.D. degrees in computer science from Washington State University, USA, in 1986 and 1990, respectively. He is currently a Chair Professor and the Head of the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. He has co-authored five books in mobile computing and wireless sensor networks, co-edited nine books, and authored over 500 papers in major international journals and conference proceedings. His research interests include parallel and distributed computing, wireless networks and mobile computing, big data and cloud computing, pervasive computing, and fault tolerant computing. He is a member of ACM and a Senior Member of the China Computer Federation.

ZHENGYU WANG received the B.Sc. degree in computer science from Xiamen University, China, in 1987, and the M.Sc. and Ph.D. degrees in computer science from the Harbin Institute of Technology, China, in 1990 and 1993, respectively. He is currently a Professor and the Dean of the School of Software, South China University of Technology. His research interests include distributed computing and SOA, operating systems, software engineering, and large-scale application design and development.