Coflow Scheduling in the Multi-resource Environment

Jianhui Zhang, Deke Guo, *Senior Member*, *IEEE*, Keqiu Li, *Senior Member*, *IEEE*, Heng Qi, Xiaoyi Tao and Yingwei Jin

***Abstract*—In data centers, a lot of cluster computing applications follow the coflow working pattern. That is, a collection of flows between two groups of machines is semantically related. On the other hand, network function virtualization (NFV) sufficiently improves the performance of data center networks. It, however, complicates the network environment by introducing many multi-function middleboxes each with multiple resources. Coflows encounter extremely different processing delays under diverse network functions. Prior coflow scheduling schemes are insufficient to guarantee the coflow completion time (CCT) in the multi-resource environment. In this paper, we propose, model, and analyze the coflow scheduling problem in the multi-resource environment. We present a dedicated method, DRGC (Data Rate Guarantee for Coflow), to guarantee the data rate requirements of coflows in this situation. DRGC prioritizes the coflow scheduling sequence, assigns precise data rates for coflows, and deploys a packet scheduling algorithm at middleboxes to guarantee their transmissions. In our experiments, DRGC efficiently guarantees the completion times of coflows and supports 15% more workload, compared with other scheduling schemes.**

***Index Terms*—Coflow scheduling, QoS guarantee, multi-resource environment.**

I. Introduction

In data centers, cluster computing applications, e.g., MapReduce [1] and Dryad [2], have been vastly hosted. These data-intensive applications work in the job granularity. Each job contains lots of coflows, each of which can be defined as a collection of data flows between two groups of machines. Coflow is proposed to abstract *the communication requirements of prevalent data parallel programming paradigms* [3]. Although the individual flows in the same coflow are usually transmitted in parallel, the coflow completion time (CCT) is always delayed by the last finished flow in it. Meanwhile, the input of a coflow usually depends on the output of other coflows. The dependency relationship between coflows further complicates the coflow scheduling problem.

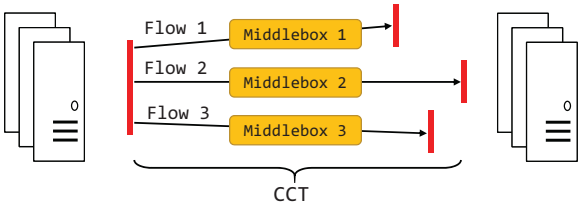


Fig. 1. Coflow scheduling with middleboxes.

Cluster applications usually have strict Quality of Service (QoS) requirements on the CCT. In web searching, the aggregator needs to integrate the feedback information from edge workers. A long feedback time of an individual flow delays the final response to users. A variety of scheduling schemes [4–6] have been proposed to improve the performance of cluster applications. However, prior works are insufficient to achieve better performance when they neglect the collective objective of the individual flows. Subsequently, the concept of coflow has been proposed to bridge the gap. CCT is closely related to the data rates of the flows in a coflow. Thus, some rate-based coflow scheduling schemes emerged to guarantee the CCT through different ways. Typically, Baraat schedules coflows in a multiplexed FIFO (First In First Out) manner [7]. Varys measures the expected CCT and preferably schedules the coflow with the minimal CCT [8]. RAPIER strives to minimize the CCT through an appropriate routing scheme [9]. All of these schemes assume that the individual flows only encounter simple forwarding at network devices. Thus, data packets experience roughly the same processing delay at these devices. However, these assumptions are difficult to achieve in networks deployed with multi-function middleboxes.

Network function devices, also known as middleboxes [10], have been ubiquitously deployed in data centers, on par with traditional L2/L3 network devices [11, 12]. Customized middleboxes incur high cost, including the purchase and the upgradation of the physical equipments [13]. Network function virtualization (NFV) sufficiently reduces the cost by deploying various network functions on commodity hardware. Thus, NFV devices are also seen as software-centric middleboxes [10]. Some noticeable characteristics of middleboxes further complicate the coflow scheduling problem in data centers.

A coflow contains multiple data flows in parallel [8]. These flows usually experience different transmission delays on their

This work is partially supported by the State Key Program of National Natural Science of China (Grant No. 61432002); NSFC (Grant Nos. 61772112, 61672379, 61602226, 61751203, and 61772544); the Dalian High-level Talent Innovation Program (No. 2015R049), and the Hunan Provincial Natural Science Fund for Distinguished Young Scholars (Grant No. 2016JJ1002).

Corresponding authors: Deke Guo, Keqiu Li.

Jianhui Zhang is with the School of Information and Security Engineering, Zhongnan University of Economics and Law, Wuhan, 430073, China. E-mail: zhangjh@zuel.edu.cn.

Deke Guo is with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, Hunan, 410073, P. R. China. He is also with the College of Intelligence and Computing, Tianjin University, Tianjin, 300350, P. R. China. E-mail: dekeguo@nudt.edu.cn.

Keqiu Li, Heng Qi and Xiaoyi Tao are with the School of Computer Science and Technology, Dalian University of Technology, Dalian, 116024, China. E-mail: likeqiu@gmail.com, hengqi@dlut.edu.cn, taoxiaoyi@mail.dlut.edu.cn.

Yingwei Jin is with the School of Management, Dalian University of Technology, Dalian, 116024, China. E-mail: jinyw67@dlut.edu.cn.

respective routing paths [9], which are deployed with diverse middleboxes. In a special case, some applications are deployed and executed across geographically distributed data centers [14]. In this situation, a coflow deriving from these applications needs to go through diverse middleboxes and undergo the same or different network functions in different data centers. With different workloads and hardware configurations, it is hard to synchronize the transmissions of the flows in the same coflow, which means its completion time cannot be guaranteed. The reasons are as follows. Compared with routers and switches, middleboxes perform a variety of network functions, including firewall, IDS, basic forwarding, and so on. Typically, security-based packet analyzing functions [15] consume more CPU cycles than other functions. For a packet with the size of 1000 bytes, Redundancy Elimination and IPSec Encryption respectively consume almost $2\times$ and $11\times$ of the CPU time, compared with the basic packet forwarding [16]. As shown in Fig. 1, flows 1, 2 and 3 are set with the same size and they belong to the same coflow. When flow 1 is simply forwarded at middlebox 1, the transmissions of flows 2 and 3 will extremely delay the completion of this coflow if they undergo Redundancy Elimination and IPSec Encryption, respectively. Even if these middleboxes execute the same network function, different workloads at these devices will still result in transmission discrepancy. However, the transmissions of the individual flows should be completed simultaneously such that none of them will delay the completion of the coflow.

Meanwhile, data rate partition among flows is difficult in the multi-resource environment [16]. Network devices, including switches, routers and middleboxes, are equipped with multiple hardware resources, e.g., the CPU, the memory, and the NIC. Packets need to be transferred to the next resource after passing the previous one [17][18]. However, packet forwarding at switches and routers consumes short processing time at the CPU and the memory, and only the NIC will restrict the transmissions of flows [16]. In this situation, the bandwidth of the NIC can be discretionarily allocated to flows without concern for the capabilities of the CPU and the memory. As we mentioned before, flows under different network functions consume different amounts of resources at middleboxes. Thus, any resource may become the bottleneck. Focusing on the data rate partition only on one resource may exhaust other resources. Fairly allocating each resource in a middlebox will not achieve a fair data rate partition for the passing flows [16]. Not to mention that some hardware resources cannot be divided or shared by packets simultaneously. Thus, traditional rate-based coflow scheduling schemes become inapplicable in the multi-resource environment. Here, we do not consider the packet processing in the systems with multiple CPUs or NICs.

In this paper, we propose, model, and analyze the coflow scheduling problem in the multi-resource environment. We present DRGC (Data Rate Guarantee for Coflow) to guarantee the completion times of coflows in the multi-resource environment. DRGC first determines the coflow scheduling sequence and prioritizes their transmissions. The CCTs are closely related to the data rates of the individual flows, thus DRGC makes precise data rate allocation for the individual flows such that their transmissions can be completed simultaneously before the predefined completion times. We incorporate endpoints and middleboxes together to achieve this goal. Endpoints expose the desired data rates of flows through timestamp marking on packets. Middleboxes adopt a scheduling algorithm to satisfy their requirements on data rates, based on the priorities of coflows. In our experiments, DRGC efficiently guarantees the completion times of coflows and supports 15% more workload, compared with other scheduling schemes.

The rest of this paper is organized as follows. We introduce related scheduling schemes in §II. The background knowledge about coflow and the motivation of DRGC are given in §III. Detailed framework of DRGC is introduced in §IV. We conduct extensive trace-driven experiments to verify the performance of DRGC in §V. §VI concludes this paper.

II. RELATED WORK

Scheduling schemes can be classified into many categories based on different criterions. In this part, we introduce some coflow-aware scheduling schemes and some schemes used in the multi-resource environment.

A. Coflow-aware Scheduling Schemes

Prior works [4–6] have already attempted to improve the performance of cluster computing applications. Formally, coflow [3] represents the application-level semantics among individual flows, and many coflow-aware scheduling schemes have been successively proposed to guarantee the communication requirements of coflows. As a task-aware scheduling scheme, Baraat [7] schedules coflows in a FIFO manner. It detects large coflows and then increases the level of multiplexing, so as to avoid the head-of-line blocking. Thus, Baraat possesses the characteristics of decentralization and non-preemption. On the contrary, Varys [8] implements in a centralized and preemptive manner. It computes the minimum CCT for coflows, and uses the proposed Smallest-Effective-Bottleneck-First (SEBF) algorithm to schedule coflows in the smallest-CCT-first order. However, Varys suffers from the scalability problem. For avoiding the aforementioned limitations of Baraat and Varys, D-CAS [19] attempts to minimize the average CCT through a decentralized and preemptive manner. Periodically, senders of each subcoflow negotiate with the receivers about their desired priorities. Then, senders update priorities of packets according to the feedback from receivers. Subcoflow-level minimum-remaining-time-first is achieved through priority-based scheduling policy at switches.

RAPIER [9] additionally takes the routing scheme into account and appropriately selects paths for the individual flows of each coflow, so as to minimize its CCT. Subsequently, it allocates minimal bandwidth to flows to synchronize their completion time and distributes the remaining bandwidth to other coflows. As a flow information-agnostic scheduling scheme, Aalo [20] implements a least-attained service scheme by using a multi-priority queueing system. The coflows with the same priority will be allocated to a common queue and be scheduled in the FIFO manner. Besides, the priority of each coflow decreases according to the traffic it has already sent. Thus, the coflows with small size are more likely to be finished

before larger ones, and the average CCT can be reduced. Specially, Sunflow [21] explores the coflow scheduling problem in the network deployed with optical circuit switches.

Prior works neglect the packet processing procedure at middleboxes, which are equipped with multiple hardware resources. Flows consume more processing time at these devices, compared with routers and switches. The benefit of coflows is difficult to be guaranteed when the individual flows of the same coflow pass through different middleboxes. We propose DRGC to guarantee the data rate requirements of coflows in this situation. This is the most important difference between our proposal and prior works.

B. Flow Scheduling in the Multi-resource Environment

Traditional queueing schemes strive to guarantee the QoS requirements of flows in a single resource queueing system. For example, Fair Queueing (FQ) [22], Weighted Fair Queueing (WFQ) [23], Start-time Fair Queueing (SFQ) [24] and Generalized Processor Sharing (GPS) [25] strive to provide fair service for the passing flows. In the multi-resource environment, flows usually consume different amounts of resources under diverse network functions. Meanwhile, incoming packets need to be successively processed on diverse resources. Traditional flow scheduling schemes become inefficient to guarantee the QoS requirements of flows under these constraints. Dominant Resource Fairness (DRF) [26] presents a fair multi-resource allocation principle for users with diverse demands on different resources. Here, the dominant resource of a user is defined as the resource with the maximum share among all resources. Inspired by DRF, many scheduling schemes [16][17][18][27][28] have been proposed to provide fair service in the multi-resource environment. Subsequent works further improve the availability of this kind of scheduling scheme in many aspects.

Typically, Multi-Resource Fair Queueing (DRFQ) [16] applies virtual time to measure each flow's processing time on its dominant resource. According to this metric, flows are scheduled in a max-min fairness manner, i.e., the flow with the minimum virtual time is always scheduled preferably. In this way, the DRF among flows is achieved in time, rather than in space. Similarly, Dominant Resource Generalized Processor Sharing (DRGPS) [27] introduces the concept of GPS into the multi-resource environment and achieves DRF for flows strictly at all time. Although DRGPS can only be realized in an ideal fluid model, where packets can be subdivided infinitely, it should be taken as a benchmark to evaluate the performance of other scheduling schemes.

Prior scheduling schemes suffer from the scalability problem when too many flows are backlogged in the system. Thus, Multi-Resource Round Robin (MR^3) [17] proposes to reduce the computing complexity by using the round-robin algorithm. In each scheduling round, the flow at the head of the list of active flows will be chosen. Then, it can optionally process its packets as long as its balance, which is reduced according to the dominant processing time of packets, is positive. Compared with prior works, MR^3 only need $O(1)$ time cost to make scheduling decisions while still target at achieving the DRF among flows. However, MR^3 may result in large scheduling delay when flows are assigned with different weights. GMR^3 [18] eliminates this concern by separating the flows with similar weights into different groups, each of which associates with a timestamp. The flows in the group with the minimum timestamp will be scheduled in a round-robin manner. Besides the fairness, ATFQ [29] also strives to improve the resource utilization by using an efficient scheduling algorithm.

All the aforementioned scheduling schemes deploy per-flow buffers for flows. This results in more scheduling overhead with the growing number of the arrived flows. To solve this problem, Myopia [10] presents an improved count-min sketch to identify elephant flows. Only these flows will be allocated a per-flow buffer and scheduled subjecting to the DRF. Meanwhile, the mice flows will be buffered in a common queue and follow the FIFO scheduling order. Myopia is effective because it only maintains the state of a few elephant flows, which contribute the majority of the traffic load in the network.

III. COFLOW SCHEDULING IN THE MULTI-RESOURCE ENVIRONMENT

We start with the concept of coflow in §III-A. Then, we model the coflow scheduling problem in the multi-resource environment in §III-B. For protecting the benefit of coflows, coflow scheduling priorities are determined in §III-C. Finally, we make precise data rate allocation for the individual flows of coflows, so as to guarantee their predefined completion times.

A. Problem Background

A coflow is defined as *a semantically-related collection of flows between two groups of machines* [3]. The individual flows in a coflow derive from the same group of source nodes, and target at the same group of destination nodes. With a collective objective, their transmissions are usually in parallel. For example, web search works in the Partition/Aggregate pattern [30]. After analyzing the searching request from the aggregator, all the end-workers will return the searching results to the aggregator. These parallel flows are semantically-related, because they all contribute to the final response to users. Thus, they can be seen as a single coflow. In this context, the collective objectives of coflows should be taken into account. Generally, coflow represents many communication modes, e.g., many to one, one to many, and all to all, depending on the work patterns of different applications. Here, we list some necessary notations in Table I.

Normally, a coflow c_i can be expressed as a collection of individual flows: $c_i=\{f_{i,1}, f_{i,2}, ..., f_{i,|c_i|}\}$. $|c_i|$, which is also defined as the width of c_i, indicates the number of flows in this coflow. As prior works [9][31], we assume that coflow information, including the sizes of individual flows, is achievable by using some prediction techniques [32]. The size of a coflow indicates the sum of sizes of individual flows in it. As for the flow $f_{i,j}$, its size and data rate are denoted as $size(f_{i,j})$ and $r_{i,j}$, respectively. Thus, the size of the coflow c_i can be calculated as:

$$size(c_i)=\sum_{j=1}^{|c_i|} size(f_{i,j}). \tag{1}$$

TABLE I
NOTATIONS.

Notation	Explanation
$size(p)$	the size of packet p
$size(f_{i,j})$	the size of flow $f_{i,j}$
$start(c_i)$	the start time of coflow c_i
$end(c_i)$	the actual completion time of coflow c_i
$start(f_{i,j})$	the start time of flow $f_{i,j}$
$end(f_{i,j})$	the completion time of flow $f_{i,j}$
d_i	the predefined completion time of c_i
$r_{i,j}$	the data rate of flow $f_{i,j}$
$s_{i,j}$	the completion status of flow $f_{i,j}$
$\mathbf{R}_{\widetilde{m}}$	the $\widetilde{m}$-th resource at a middlebox
$\mathcal{S}(p,\widetilde{m})$	the start time of packet p on $\mathbf{R}_{\widetilde{m}}$
$\mathcal{L}(p,\widetilde{m})$	the processing time of packet p on $\mathbf{R}_{\widetilde{m}}$
$\mathcal{R}(p,\widetilde{m})$	the release time of packet p on $\mathbf{R}_{\widetilde{m}}$

If the flow $f_{i,j}$ starts its transmission at $start(f_{i,j})$, its completion time is given as follows:

$$end(f_{i,j}){=}start(f_{i,j}){+}\frac{size(f_{i,j})}{r_{i,j}}. \tag{2}$$

The individual flows in c_i transmit their data in parallel, but out of synchronization. The first started flow and the last finished flow respectively indicate the start time and the completion time of the coflow, which can be expressed as:

$$start(c_i){=}\min_{f_{i,j}} start(f_{i,j}) \tag{3}$$

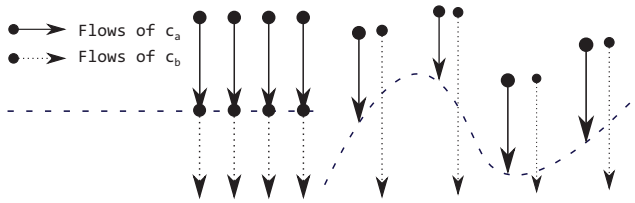
$$end(c_i){=}\max_{f_{i,j}} end(f_{i,j}). \tag{4}$$

As for some cluster computing applications, e.g., Dryad [2] and Spark [33], their jobs usually cover multiple stages. In each stage, several coflows need to be transmitted in parallel. Besides, the execution of a coflow may depend on the output of others in the previous stage. This inherent relationship plays an important role when deciding the coflow scheduling sequence in practice. Similar to [3][20], we define two kinds of dependency relationship among coflows as:

- Start-after ($c_a \nrightarrow c_b$): In Fig. 2(a), the transmission of c_b relies on the whole output of c_a. Thus, c_b can only start after c_a finishing its transmission.
- Finish-after ($c_a \rightarrow c_b$): In Fig. 2(b), the transmission of c_b depends partially on the output of c_a. Although c_b can start concurrently with c_a, it can only finish after c_a finishing its transmission. For example, MapReduce Online [34] makes it possible to push data from map task to reducers as it is produced. Thus, reducers can start their work before the completion of the map tasks.

B. Problem Modeling

Scheduling Motivation Given a set of coflows $\{c_1, c_2, ..., c_k\}$, we assume all of these coflows pass through the same one middlebox. Their completion times are predefined as $\{d_1, d_2, ..., d_k\}$. The benefit of coflows will not be influenced if they successfully finish their transmissions before their predefined completion times. In this context, fair sharing is powerless to guarantee the benefit of coflows when they compete for the limited resources at this middlebox.



(a) With barrier. (b) Without barrier.

Fig. 2. Dependency among coflows.

Striving to satisfy all the transmission requirements of coflows will result in inefficiency, on the contrary. Reasonably, the benefit of coflows should be sequentially guaranteed. In the essence, the benefit of each coflow relies on the completion of the individual flows in it. Assume that the coflow c_i is currently scheduled and it possesses the highest priority. For furthest completing its transmission, we strive to maximize the number of the individual flows meeting the predefined completion time. Thus, we get:

$$\begin{aligned} &Max \sum_{j=1}^{|c_i|} s_{i,j} \\ &s.t.\ s_{i,j} = \begin{cases} 0, & if\ end(f_{i,j}) > d_i \\ 1, & if\ end(f_{i,j}) \leqslant d_i \end{cases} \end{aligned} \tag{5}$$

Here, $s_{i,j}$ denotes the completion status of the flow $f_{i,j}$. $end(f_{i,j})$ depends on the finish time of its last packet. Thus, the packet scheduling manner at middleboxes sufficiently influences the scheduling results.

Packet Processing inside Middleboxes Assume there are m kinds of resources, denoted as $\mathbf{R}_1$, $\mathbf{R}_2$, ..., and $\mathbf{R}_m$. We make the following constraints:

- Every packet should follow the same resource processing sequence, i.e., all packets should orderly go through $\mathbf{R}_1$, $\mathbf{R}_2$, ..., and $\mathbf{R}_m$.
- The resource will be monopolized by the packet processed on it. Before finishing the processing, that resource will not be utilized to process another packet.
- For limiting the influence of the buffer system, we assume that any resource can only buffer one packet. That is, when a packet has been finished on one resource, it will be pushed to the next resource as long as the previous packet has already released that resource.

According to these constraints, the aforementioned scheduling objective directly depends on the packet scheduling sequence, denoted as η, at the middlebox. η contains all the packets of flows coexisting with the coflow c_i. With different η, flows will complete their transmissions in different sequences. The completion status of c_i also relies on η. This relationship can be expressed as follows:

$$\sum_{j=1}^{|c_i|} s_{i,j}{=}\varphi(\eta). \tag{6}$$

Consequently, we strive to get the packet scheduling sequence

η^* in which c_i can furthest completes its transmission:

$$\begin{aligned} \eta^* &= arg\ max\ \varphi(\eta) \\ s.t.\ \mathcal{R}(p^k,\widetilde{m}) &= Max\Big\{\mathcal{R}(p^{k-1},\widetilde{m}+1)\ ,\ \mathcal{S}(p^k,\widetilde{m})+\mathcal{L}(p^k,\widetilde{m})\Big\} \\ \mathcal{S}(p^k,&\widetilde{m}+1)=\mathcal{R}(p^k,\widetilde{m}),\ 1\leqslant\widetilde{m}<m \\ s_{i,j}&=\begin{cases} 0,\ if\ \mathcal{R}(p^*_{i,j},m) > d_i \\ 1,\ if\ \mathcal{R}(p^*_{i,j},m) \leqslant d_i \end{cases} \end{aligned} \tag{7}$$

We denote the k^{th} packet in the scheduling sequence η^* as p^k. Its start time, the processing time, and the release time on the resource $\mathbf{R}_{\widetilde{m}}$ are denoted as $\mathcal{S}(p^k,\widetilde{m})$, $\mathcal{L}(p^k,\widetilde{m})$, and $\mathcal{R}(p^k,\widetilde{m})$, respectively. If p^k has finished its process on $\mathbf{R}_{\widetilde{m}}$, but $\mathbf{R}_{\widetilde{m}+1}$ is still occupied by the previous packet, p^k will be buffered on $\mathbf{R}_{\widetilde{m}}$ until $\mathbf{R}_{\widetilde{m}+1}$ becomes idle. Thus, we get:

$$\mathcal{R}(p^k,\widetilde{m})=\mathcal{R}(p^{k-1},\widetilde{m}+1). \tag{8}$$

Otherwise, p^k will release $\mathbf{R}_{\widetilde{m}}$ after finishing its process on it. In this situation, we get:

$$\mathcal{R}(p^k,\widetilde{m})=\mathcal{S}(p^k,\widetilde{m})+\mathcal{L}(p^k,\widetilde{m}). \tag{9}$$

p^k will also start its process on $\mathbf{R}_{\widetilde{m}+1}$ after releasing $\mathbf{R}_{\widetilde{m}}$. Obviously, packets will release the last resource $\mathbf{R}_m$ as long as they finish their process on it. Denote the last packet of the flow $f_{i,j}$ as $p^*_{i,j}$. $f_{i,j}$ successfully completes its transmission if $\mathcal{R}(p^*_{i,j},m)\leqslant d_i$.

In the scenario of multiple middleboxes, the flows in the same coflow pass through different middleboxes. Coflow transmission becomes more complex in this situation. The biggest obstacle to the aforementioned scheduling objective is that different middleboxes do not simultaneously see c_i as the coflow whose benefit should be guaranteed at present. Otherwise, they can protect the transmission of c_i based on local knowledge. To this end, two kinds of solutions can be adopted. On one hand, middleboxes can cooperate with each other. With global knowledge, it is easy for middleboxes to determine which coflow should be scheduled currently. However, this solution results in additional communication overhead and implementation complexity. On the other hand, coflows should be distinguished by using the priority. If c_i is currently scheduled and possesses the highest priority, all middleboxes will preferentially guarantee its transmission. Prioritizing the coflow scheduling sequence is significant to achieve our scheduling objective when network resources are scarce. Next, we elaborate how to assign priorities to coflows.

C. Coflow Scheduling Sequence

Cluster computing applications usually work at the job granularity, and each job consists of multiple coflows. Although we can prefer the jobs of a specific application by allocating higher priorities to them, we do not want to discriminate against any jobs. Meanwhile, it is hard to estimate the importance of a job just according to its execution time or data volume. In this situation, First Come First Serve (FCFS) comes out to be a reasonable choice. FCFS sequentially schedules jobs just according to their arising times, thus it does not distinguish jobs in terms of the execution time or the type of application. Consequently, we decide to schedule jobs in a FCFS sequence in the design of DRGC. However, DRGC does not strictly schedule jobs one after another. We just strive to preferably satisfy the transmission requirement of the first emerged job. After that, the residual network resources will still be utilized to serve other jobs.

Network devices make scheduling decisions at the flow level, thus they are insufficient to distinguish different coflows. For identifying coflows, we make the following rules. Job J_N registers the number N on a global counter, which increases with the registration of jobs. Obviously, the value of N is unique, and it records the job registration sequence, which depends on the job arising time. Meanwhile, the execution of J_N usually covers multiple transmitting or computing stages, each of which contains multiple parallel coflows. We use s and n to indicate the serial number of the stage and the serial number of the coflow in each stage, respectively. j is used to indicate the serial number of each individual flow in the same coflow. Based on these rules, we use $\gamma=\langle N,s,n,j\rangle$ as the indicator of each individual flow. The reasons are as follows.

1) γ is unique in the network. Different jobs will not be assigned the same value of N. Meanwhile, cluster computing applications can assign $\langle s,n,j\rangle$ to individual flows at their own end-points after registering the number N on the global counter. But in each job, values of $\langle s,n,j\rangle$ are also unique for each individual flow. Consequently, individual flows in the whole network will be distinguished from each other by using the indicator $\langle N,s,n,j\rangle$.

2) The semantic relationship between the individual flows of the same coflow can be expressed. Flows with the same values of $\langle N,s,n\rangle$ will be recognized as the ones deriving from the same coflow. Thus, they will be treated equally at different devices.

3) Coflow scheduling sequence can be defined. As aforementioned, job J_N registers the number N on a global counter. That means, jobs with earlier arising times will be assigned smaller value of N. We schedule jobs based on FCFS, thus smaller value of N indicates higher priorities in the design of DRGC. In each job, the input of coflows depends partially or totally on the output of other coflows in the previous stage. That means, coflows in the next stage will not complete their transmissions before the completion of the coflows in the previous stage. Thus, transmission failures of the latter can even interrupt the transmission of the former. For avoiding such situation, we preferentially guarantee the transmissions of the coflows in the previous stage and define that smaller value of s indicates higher priorities at stage level. Parallel coflows in the same stage should be seen as independent units. Following the principles of FCFS, we define that coflows with smaller value of n should be preferentially scheduled. Similarly, individual flows with smaller value of j in the same coflow possess higher priorities.

In summary, we use $\gamma=\langle N,s,n,j\rangle$ as the indicator of each individual flow. Meanwhile, smaller values of $N,s,n,$ and j indicate higher priorities at different levels. As for two flows with indicators of $\langle N_1,s_1,n_1,j_1\rangle$ and $\langle N_2,s_2,n_2,j_2\rangle$, we firstly schedule the former if any one of the following items is matched:

- $N_1 < N_2$;
- $N_1 = N_2$ & $s_1 < s_2$;
- $N_1 = N_2$ & $s_1 = s_2$ & $n_1 < n_2$;
- $N_1 = N_2$ & $s_1 = s_2$ & $n_1 = n_2$ & $j_1 < j_2$;

We propose DRGC to guarantee the predefined completion times of coflows in the multi-resource environment. It still works even if coflows are scheduled in other sequences. Shortest Job First (SJF) can also be adopted to achieve the corresponding scheduling objective, but it needs to measure the transmission time of coflows in advance. Now that we have already decided the coflow scheduling sequence, how to guarantee the predefined completion times of coflows becomes the most difficult challenge. Bear in mind that CCT is closely related to the data rates of individual flows. For achieving this goal, we need to make precise data rate allocation for coflows.

D. Data Rate Requirements of Coflows

Individual flows in the same coflow are semantically related. Thus, they should be selfless to achieve the collective objective of the coflow. Prior works have already explored many possible ways to minimize the average CCT in traditional networks. In another way, we predefine the CCTs of coflows in advance. Coflows coming from diverse applications undergo different data processing. Thus, uniformly predefining the CCTs for all coflows is unacceptable in practice. Different applications should determine CCTs for their coflows independently, based on statistical analysis and measurement.

As for the coflow c_i, its completion time depends on the last finished flow in it. Allocating more bandwidth to other flows will not reduce its CCT. For avoiding resource waste on the link bandwidth, other flows should reduce their data rates so as to lengthen their transmission times and finish simultaneously with the last finished flow. In this way, a coflow can complete its transmission with the minimum link bandwidth occupation, and more coflows can share the network simultaneously. Thus, we predefine all the completion times of individual flows in c_i as d_i. Given the sizes and start times of individual flows, the data rate of $f_{i,j}$ can be expressed as:

$$r_{i,j} = \frac{size(f_{i,j})}{d_i - start(f_{i,j})}. \tag{10}$$

Additionally, when there exists *finish-after* dependency between two flows of successive coflows, denoted as $f_{i',j'} \rightarrow f_{i'',j''}$, $f_{i'',j''}$ can start as long as $f_{i',j'}$ starts. Thus, $size(f_{i'',j''})$ cannot be confirmed in advance, because it depends on the output of $f_{i',j'}$. The output of $f_{i',j'}$ is not equal to its original size after data processing. Conservatively, we assume the total output of $f_{i',j'}$ linearly depends on the size of $f_{i',j'}$. Thus, the input of $f_{i'',j''}$ is:

$$size(f_{i'',j''}) = \tau \cdot size(f_{i',j'}). \tag{11}$$

The value of τ depends on the corresponding applications. If $f_{i'',j''}$ starts its transmission also at $start(f_{i',j'})$, we get its data rate as:

$$\begin{aligned} r_{i'',j''} &= \frac{\tau \cdot size(f_{i',j'})}{d_{i''} - start(f_{i',j'})} \\ &= \frac{\tau \cdot \left(d_{i'} - start(f_{i',j'})\right)}{d_{i''} - start(f_{i',j'})} \cdot r_{i',j'} \\ &= \tau^* \cdot r_{i',j'}, \end{aligned} \tag{12}$$

where $\tau^* = \tau \cdot \frac{d_{i'} - start(f_{i',j'})}{d_{i''} - start(f_{i',j'})}$. Obviously, $r_{i'',j''}$ also linearly depends on $r_{i',j'}$. The benefit of doing so is to simultaneously transmit the flows within the relationship of *finish-after*. Thus, the transmission of $f_{i'',j''}$ will not be blocked after that of $f_{i',j'}$. If the value of τ is inappropriately selected, $r_{i'',j''}$ can also be precisely recomputed after $f_{i',j'}$ finishing its transmission. After determining the data rates of individual flows in each coflow, we will introduce the design of DRGC.

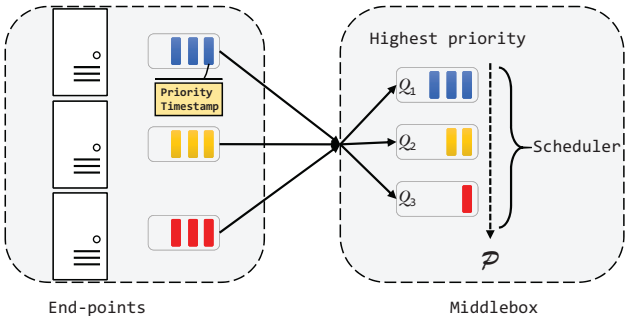


Fig. 3. Scheduling framework.

IV. Design of DRGC Scheduling Framework

We attempt to guarantee the predefined completion times of coflows based only on their priorities at middleboxes. However, this results in loss on the performance. The reason is that middleboxes are agnostic to the exact data rate requirements of flows. The flows with higher priorities occupy too many resources at middleboxes, and leave few scheduling opportunities to other flows. We propose DRGC to avoid all of these drawbacks. Its implementation covers the components at end-points and middleboxes together. End-points indicate the data rate requirements of coflows by using the timestamp marking, which, as well as the priorities of flows, is utilized by middleboxes to make scheduling decisions. Coflows with higher priorities can only achieve their desired data rates, which are indicated by the timestamps attached on packets. Thus, they cannot encroach too many resources at middleboxes. Meanwhile, other coflows will not be blocked behind the ones with higher priorities, because they can utilize the residual resources.

A. Requirement Statement about Data Rates at End-points

Now that we have already decided the coflow scheduling sequence and assigned precise data rates for individual flows, scheduling algorithms need to be designed to guarantee their data rate requirements at middleboxes. We use Fig. 3 as the guide of our scheduling framework. Data rate partition is difficult in the multi-resource environment. Flows undergoing

different network functions consume different amounts of resources, and packets get unequal processing rates on different resources. As aforementioned, if we want to maintain the desired data rates of flows at middleboxes, end-points should expose the demands of coflows to these devices. In this part, we focus on expressing the data rate requirements of coflows by using timestamp.

We denote the sending rate of flow f_γ as r_γ. If f_γ starts its transmission at $start(f_\gamma)$, the traffic that it has already sent, denoted as $\Gamma(f_\gamma, t)$, at time t can be expressed as:

$$\Gamma(f_\gamma, t) = (t - start(f_\gamma)) \cdot r_\gamma. \tag{13}$$

At the packet level, f_γ can be seen as a collection of packets, denoted as $\{p_\gamma^1, p_\gamma^2, ..., p_\gamma^{|k|}\}$. Here $|k|$ indicates the number of packets belonging to it. If we denote the sending time sequence of these packets as $\mathcal{T} = \{t_1, t_2, \ldots, t_{|k|}\}$, we get:

$$\begin{cases} t_1 = \frac{size(p_\gamma^1)}{r_\gamma} + start(f_\gamma) \\ t_{\widehat{k}} = \frac{\sum_{i=1}^{\widehat{k}} size(p_\gamma^i)}{r_\gamma} + start(f_\gamma), \quad 1 < \widehat{k} \leqslant |k| \end{cases} \tag{14}$$

According to Eq. 14, we get:

$$r_\gamma = \frac{\sum_{i=1}^{\widehat{k}} size(p_\gamma^i)}{t_{\widehat{k}} - start(f_\gamma)}. \tag{15}$$

Reasonably, we can estimate the data rate of a flow depending on some necessary information, including the start time of the flow, the sizes of its packets and $t_{\widehat{k}}$, in middleboxes. However, this will inevitably result in enormous overhead of information maintenance, especially when the number of flows passing through the middlebox is huge. To alleviate this concern, we should refine the information implied in $\mathcal{T}$. Based on Eq. 14, we get:

$$t_{\widehat{k}} - t_{\widehat{k}-1} = \frac{size(p_\gamma^{\widehat{k}})}{r_\gamma}, \quad 1 < \widehat{k} \leqslant |k| \tag{16}$$

Thus, the time intervals between the time sequence in $\mathcal{T}$ have no relationship with $start(f_\gamma)$. For decoupling $start(f_\gamma)$ from t_1, we further adjust $\mathcal{T}$ as:

$$\begin{aligned} \mathcal{T} &= \mathcal{T} - t_1 \\ &= \{t_1 - t_1, t_2 - t_1, \ldots, t_{|k|} - t_1\} \\ &= \{t_{1'}, t_{2'}, \ldots, t_{|k|'}\}. \end{aligned} \tag{17}$$

Finally, we use $\mathcal{T}$ as the timestamps of packets to implicate the footprint of f_γ with the sending rate of r_γ. $p_\gamma^{\widehat{k}}$ will be attached two tags, i.e., γ and $t_{\widehat{k}}$, when it enters the network. A new flow can be easily recognized when p_γ^1 with the timestamp of $t_{1'}$=0 arrives at the middlebox. We also set $t_{|k|'} = -1$ to indicate the last packet of a flow. In addition, the retransmitted packets will be assigned the same timestamps as before. Thus, they will not influence the timestamp computation of other packets. Next, we will explain how to satisfy the data rate requirements of flows at middleboxes.

Algorithm 1 Preprocessing

Require: $\mathcal{P}$, the list of flow information. $\mathcal{Q}_\gamma$, the buffer list of f_γ. $\mathcal{C}$, the current system clock. $\mathcal{C}_1$, the system clock when the first packet of a flow arrives. f_γ, data flow with the priority of γ. p_γ, any packet of f_γ. ts, the timestamp of a packet. TS, the ideal scheduling time of a packet.

```
if p_γ.ts = 0 then
    Allocate Q_γ to f_γ;
    p_γ.TS = C;
    C_1 = p_γ.TS;
    Insert p_γ into Q_γ;
    P.insert(γ);
else if p_γ.ts = −1 then
    Insert p_γ into Q_γ;
else
    p_γ.TS = p_γ.ts + C_1
    Insert p_γ into Q_γ;
end if
```

B. Scheduling Procedure at Middleboxes

As aforementioned, end-points mark flows with the priority and timestamp. Next we strive to maintain the desired data rates of flows at middleboxes. Before that, sorting the flows according to their priorities, and synchronizing the timestamp of packets with the current system clock are necessary preparatory work for reducing the computing complexity of the scheduler.

Flow Sorting For providing distinguishing services, all the newly arrived flows should be sorted according to their indicators. Two different flows will never have the same value of γ because each γ is unique in the whole network. We adopt a list $\mathcal{P}$ to maintain the indicator information of flows in a monotone decreasing order, according to the numbers of $\langle N, s, n, j\rangle$. As aforementioned, smaller values of N, s, n, and j indicate higher priorities at different levels. Thus, each flow will be assigned an index number of the list $\mathcal{P}$. Obviously, the flow with the index of 1 possesses the highest priority. For the management of the list $\mathcal{P}$, we also define some list operations as follows:

- get (index): return the indicator of a flow according to the index number
- insert (γ): insert the indicator γ of f_γ into $\mathcal{P}$
- delete (γ): delete the information of f_γ from $\mathcal{P}$

As explained in Alg. 1, when a packet p_γ^k arrives at the middlebox, it will be checked whether f_γ is a newly arrived flow. As aforementioned, each packet of flows has been attached two tags, i.e., γ and $t_{k'}$, when it enters the network. We also predefine the timestamps of the first packet and the last packet of a flow as 0 and -1, respectively. Here we denote the timestamp of a packet as ts. If $p_\gamma^k.ts$=0, f_γ will be recognized as a new flow and get a dedicated buffer list, denoted as $\mathcal{Q}_\gamma$. Then, p_γ^k will be pushed into $\mathcal{Q}_\gamma$. In this list, packets will be sequentially buffered according to their timestamps, and the packet with the minimum timestamp will be buffered at the head of the list. Reasonably, only the packet at the head will

be checked whether it should be scheduled in one scheduling loop. Here, $|\mathcal{Q}_\gamma|$ indicates the number of packets in this list. On the contrary, if $p_\gamma^k.ts{\neq}0$, i.e., f_γ has already arrived, p_γ^k will be inserted into $\mathcal{Q}_\gamma$ at the right place. We use the buffer list $\mathcal{Q}_\gamma$ to support packet retransmission. The retransmitted packets will still be buffered at the forepart of $\mathcal{Q}_\gamma$ such that they will be quickly scheduled before their subsequent packets at this device. Finally, we perform $insert(\gamma)$ to insert the priority information of f_γ into $\mathcal{P}$ with appropriate index after comparing γ with the priorities of other flows in $\mathcal{P}$. Thus, the overhead of flow sorting depends on the number of the flows that have already arrived. As aforementioned, at most four comparisons are needed for comparing the priority information of two flows.

In other situations, if jobs or coflows should not be scheduled with the FCFS policy, we can reenact the priority rules made in §III-C, and set higher priorities to specific coflows. Subsequently, DRGC should also sort the arriving flows, according to the new priority rules, at each middlebox. The scheduling algorithm of DRGC needs no correction, and still works on the sorted flow list $\mathcal{P}$.

Time Synchronization Before pushing the packets of f_γ into the queue $\mathcal{Q}_\gamma$, we need to synchronize the timestamps of its packets with the current system clock. We introduce the symbol $\mathcal{C}$ to denote the current system clock when we use it. A new flow f_γ can be easily recognized when p_γ^1 arrives with the timestamp of 0. Ideally, this packet expects to be scheduled immediately. The reasons are that the end-point supposes the transmission of this flow will not be blocked in the network such that it can safely complete its transmission before the predefined completion time. Thus, we use p_γ^1 to record the transmission trace of this flow at different middleboxes, and define that the first packet of each flow will be scheduled immediately after its arrival. From here, we use TS to indicate the ideal scheduling time of each packet. As for p_γ^1, we get:

$$p_\gamma^1.TS = p_\gamma^1.ts + \mathcal{C} = \mathcal{C}_1. \tag{18}$$

We denote the ideal scheduling time of p_γ^1 as $\mathcal{C}_1$, which will be maintained as a constant. Before pushing the subsequent packets of p_γ^1 into $\mathcal{Q}_\gamma$, their scheduling times can be computed as follows:

$$p_\gamma^k.TS = p_\gamma^k.ts + \mathcal{C}_1. \tag{19}$$

The reason for doing this is to keep the scheduling time intervals among packets as before. Exceptionally, we will not recompute the scheduling time of the last packet of each flow, whose timestamp is set as -1. Obviously, as for each flow, the overhead of the time synchronization directly depends on the number of the packets in it.

Time synchronization happens at every middlebox, and it will not change the original timestamps of packets. End-points predefine the completion times of coflows according to their sizes. For avoiding the waste of the link bandwidth, end-points will stop the transmissions of two kinds of flows: (1) The flows who have missed their predefined completion times during their transmissions; (2) The flows whose residual time before the predefined completion time is not enough to complete their transmissions. This can be achieved by comparing the maximal needed data rate for completing the transmission of the residual data volume with the bandwidth of the NIC at endpoints. Until now, all the preparatory operations are finished, and the scheduler starts to work.

Algorithm 2 The scheduling algorithm

Require: $\mathcal{P}$, the list of flow information. $\mathcal{Q}_\gamma$, the buffer list of f_γ. $\mathcal{Q}_\gamma.head$, the packet at the head of $\mathcal{Q}_\gamma$. $\mathcal{C}$, the current system clock. γ, the priority of data flow f_γ. p_γ, any packet of f_γ. TS, the ideal scheduling time of a packet.

```
1: if |P| = 0 then
2:    Go to line 1;
3: else if |P| = 1 then
4:    γ = P.get(1);
5:    if |Q_γ| ⩾ 1 then
6:       Schedule(Q_γ.head);
7:    else
8:       Go to line 1;
9:    end if
10: else
11:    for {j = 1; j ⩽ |P|; j++} do
12:       if j = |P| then
13:          γ = P.get(|P|);
14:          if |Q_γ| ⩾ 1 then
15:             Schedule(Q_γ.head);
16:          else
17:             Go to line 1;
18:          end if
19:       else
20:          γ = P.get(j);
21:          if {(|Q_γ| ⩾ 1)&(Q_γ.head.TS ⩽ C)} then
22:             Schedule(Q_γ.head);
23:          end if
24:       end if
25:    end for
26: end if

27: Schedule (p_γ) {
28: if p_γ.TS ≠ −1 then
29:    Process the packet p_γ;
30: else
31:    Process the packet p_γ;
32:    P.delete(γ);
33: end if
34: Go to line 1;}
```

Scheduling Procedure We design Alg. 2 to guarantee the data rate requirements of coflows at middleboxes, according to their priorities and the recomputed timestamps of their packets, comprehensively. The former is utilized to distinguish the services that different coflows should receive, and the latter is responsible for data rate limitation. As aforementioned, we define that the first packet of each flow will be scheduled immediately after its arrival. Then, the scheduler starts with a check on the length of $\mathcal{P}$, denoted as $|\mathcal{P}|$. Different branches spread depending on the value of $|\mathcal{P}|$:

(1) If there is no data flow, i.e., $|\mathcal{P}|{=}0$, the scheduler will stay idle until new flows arrive.

(2) If only one flow has registered in $\mathcal{P}$, and its packets have been buffered in its queue $\mathcal{Q}_\gamma$, i.e., $|\mathcal{Q}_\gamma|\geqslant 1$, no more timestamp comparisons are needed and the packet at the head of $\mathcal{Q}_\gamma$, denoted as $\mathcal{Q}_\gamma.head$, will be processed through the method $Schedule(p_\gamma)$. In detail, if $\mathcal{Q}_\gamma.head$ is not recognized as the last packet of flow f_γ, the packet will be processed. Otherwise, the information of f_γ will be removed from the list $\mathcal{P}$ after processing its packet. Meanwhile, the buffer space occupied by this flow will be simultaneously taken back. After that, the next scheduling loop will start. Obviously, if no packets exist in $\mathcal{Q}_\gamma$, the scheduler will still restart.

(3) If $|\mathcal{P}|>1$, the scheduler need to select one flow out of these $|\mathcal{P}|$ flows to schedule. Firstly, it estimates the flow with the highest priority in $\mathcal{P}$. If its packets have been buffered in its queue, and the timestamp of the packet at the head is smaller than the current system clock, including the last packet of this flow, this packet will be scheduled. Reasonably, if the buffer space of the current flow is empty, the next flow with a smaller priority will be estimated. As an exception, if the prior $|\mathcal{P}|-1$ flows are all un-schedulable, the last flow with the minimum priority will be scheduled as long as it has packets to be processed. If not, the scheduling algorithm will restart.

In summary, the transmissions of the flows with higher priorities will be firstly guaranteed in the scheduling procedure of DRGC. After that, the residual resources will be used to support the transmissions of other flows. This principle obeys the design motivation of DRGC.

C. Interpretations about DRGC

DRGC is designed to guarantee the predefined completion times of coflows, which have a deep relationship with the data rate, in the multi-resource environment. To this end, end-points assign timestamps to the packets of each flow so as to indicate its desired data rate. The time intervals between the timestamps of the successive packets are utilized by DRGC to satisfy the desired data rate of this flow at middleboxes. Consequently, flows with higher priorities cannot occupy excessive network resources depending only on their priorities. After satisfying the data rate requirements of these flows, the residual resources will be remained for other coflows. That is, DRGC strives to guarantee the data rate requirements of high-priority coflows. After that, it also maximizes the benefit of others.

In addition, DRGC does not rely on special architectures of middleboxes or additional flow information. As aforementioned, we assumed that any resource can only buffer one packet at middleboxes. In practice, as for an individual flow, the scheduling algorithm of DRGC strives to maintain the time intervals between the timestamps of its packets at middleboxes. In this way, its desired data rate can be satisfied at these devices. In this procedure, the scheduling algorithm determines the packet scheduling occasions according to their timestamps. Then the scheduled packet will be pushed to the first resource at the middlebox. The number of resources, the sizes of buffers and the packet processing cost at middleboxes are all needless for the scheduling algorithm of DRGC. Consequently, DRGC still works when middleboxes are deployed with large buffers or more kinds of resources. Even more, DRGC also applies to the single resource environment. However, it is difficult to directly deploy DRGC at traditional switches, which support limited number of priorities and cannot schedule packets according to their timestamps. This would be our future work.

It is worth noting that DRGC, like DRFQ, assigns separated buffer space for each individual flow. The space overhead increases with the growing numbers of the backlogged flows. Actually, network traffic is mostly contributed by a small fraction of flows with huge sizes. For simplifying the management of per-flow queues, Myopia [10] only provide efficient transmission controls for huge flows. Small flows will be served by using the reserved resources. Similar schemes can be adopted to reduce the space overhead of DRGC. Meanwhile, cluster applications should set earlier completion times for coflows to accelerate their transmissions when the network traffic load is low, such that DRGC will remain work conserving.

V. PERFORMANCE EVALUATION

We designed various experiments to estimate the performance of DRGC. The experiment setting is given in §V-A. We explore the properties of DRGC in various scheduling scenarios in §V-B. Then, we use a large-scale Hive/MapReduce trace to evaluate the performance of DRGC in §V-C. The experimental results verified that DRGC achieved better performance in many aspects, in comparison with other schemes.

A. Experiment Setting

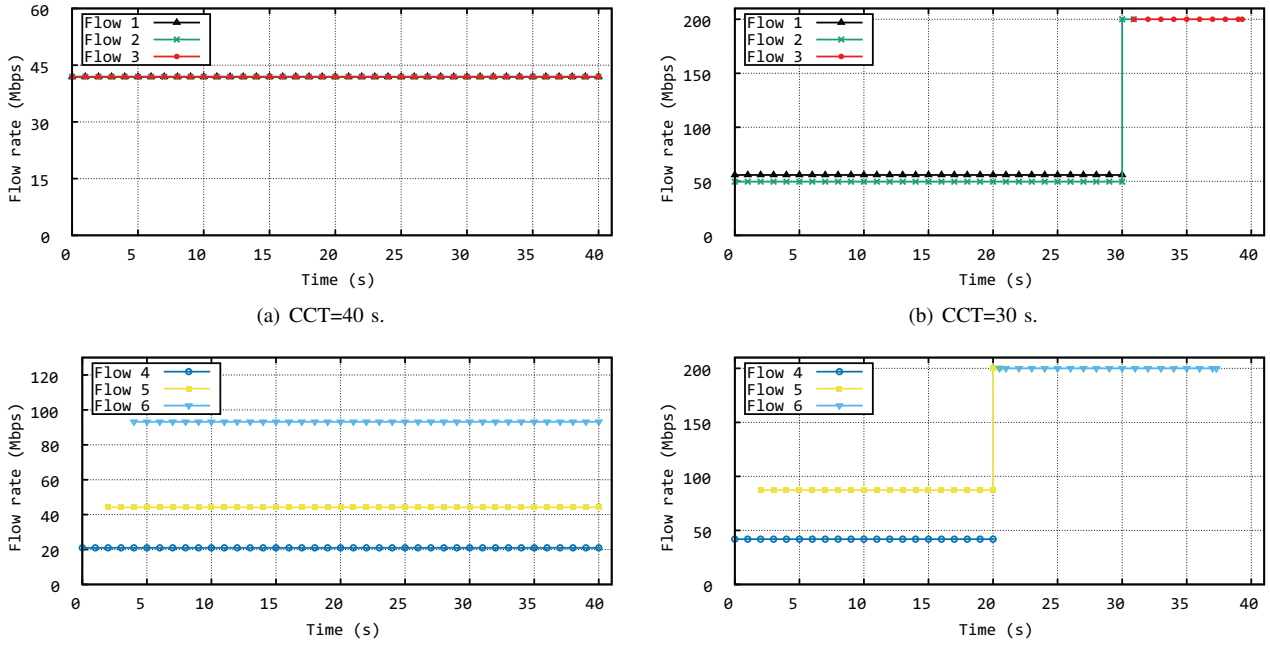
Although middleboxes perform a wide range of network functions, they possess some characteristics in common. Under some network functions, the packet processing time on a resource follows an approximate linear relationship with the packet size [16], which can be expressed as $y{=}\eta x + \xi$. Here x is the packet size in byte. η and ξ are two parameters that varies according to the performed function. We use the same CPU processing times under different functions as [17], and detailed information is listed in Table II. Here, IE, SM, BF and RE all follow this principle.

TABLE II
THE CPU PROCESSING TIME UNDER DIFFERENT FUNCTIONS.

Function	CPU Processing Time (μs)
IPSec encryption (IE)	$0.015 \cdot x + 84.5$
Statistical monitoring (SM)	$0.0008 \cdot x + 12.1$
Basic forwarding (BF)	$0.00286 \cdot x + 6.2$
Redundancy elimination (RE)	$0.00699 \cdot x + 10.97$

TABLE III
SETTINGS OF FLOWS.

Coflow ID	Flow ID	Time	Flow Size	Packet Size	Function
1	1	0 s	200 MB	600 bytes	IE
	2	0 s	200 MB	800 bytes	SM
	3	0 s	200 MB	1000 bytes	BF
2	4	0 s	100 MB	600 bytes	IE
	5	2 s	200 MB	800 bytes	SM
	6	4 s	400 MB	1000 bytes	BF



(a) CCT=40 s.

(b) CCT=30 s.

(c) CCT=40 s.

(d) CCT=20 s.

Fig. 4. Data rate partition.

In our experiments, we assume packets will be sequentially processed on the CPU and the NIC. The packet processing time on the NIC can be simply achieved according to the sizes of packets and the bandwidth of the NIC. The bandwidth of the NIC is set as 200 Mbps. Related flow information is list in Table III. The packet sizes of different flows are not set as the same such that these flows will consume more processing times on different resources. The experiments are conducted by using Java programming language with about six hundred lines of code. TCP flow transmission is simulated at the packet level, because sequenced packets reveal the data rate requirements of flows at end-points.

Bear in mind that DRGC is proposed to satisfy the predefined completion times of coflows in the multi-resource environment. CCT is closely related with the transmission rates of individual flows in each coflow. Thus, we designed three scenarios to evaluate DRGC on the aspects of satisfying the predefined CCTs, supporting stable data rates, and increasing the resource utilizations. For avoiding the influence of the congestion control in data centers, these flows will be transmitted with stable and minimal data rates, based on their predefined completion times. Meanwhile, the transmissions of the flows that have already missed the predefined completion times would continue so as to reveal the transmission control mechanism of DRGC.The used multi-function middleboxes in our experiments perform all the mentioned four kinds of functions as listed in Table II. Meanwhile, each of the passing flows will undergo just one function.

B. Properties of DRGC

Data Rate Partition In the first scenario, flows 1, 2, and 3 belong to coflow 1. They arrive at middlebox 1 simultaneously at 0 s and undergo three different functions, as listed in Table III. We firstly predefine the completion time of coflow 1 as 40 s. In Fig. 4(a), although these flows are configured with different packet sizes, their transmissions finish simultaneously at 40 s. Meanwhile, they achieve the same data rate. Obviously, when the resources at the middlebox are relatively sufficient to support the transmissions of these three flows, no flow will be blocked. In addition, all flows get stable data rates without fluctuation. This results from the design principle of DRGC that it strives to maintain the time intervals between the packets of each flow as before. Thus, data rate fluctuation can be avoided by using DRGC when the resources are sufficient.

In another case, we predefine the completion time of coflow 1 as 30 s so as to estimate the performance of DRGC in the limited resource situation. In this situation, the transmissions of these flows will exhaust the resources at the middlebox. In this part, flow transmission will continue even if some flows miss their predefined completion times. In the design of DRGC, these flows and the ones whose data rate requirements exceed the bandwidth of the NIC will be removed by endpoints. As illustrated in Fig. 4(b), flow 1, with a smaller serial number than flows 2 and 3, is firstly served with its desired data rate and finishes its transmission at the predefined completion time. Now that its transmission has not exhausted the resources, flow 2 also gets a part of its desired data rate before the completion of flow 1. After that, flow 2 finishes its transmission within another 0.9 s. Finally, flow 3 starts at 30.9 s and finishes its transmission at 39.3 s. Obviously, the data rate requirements of flows will be sequentially guaranteed by using DRGC when the resources are insufficient. Resources will be firstly utilized to service the transmission of the flow with the highest priority. After that, the residual resources will be allocated to the next flow. When resources are exhausted, the transmissions of other flows will be blocked. In this procedure, the scheduled flows will still get stable data rates until some flows release parts of resources.

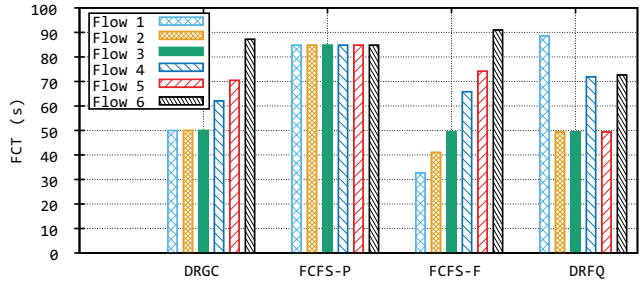


Fig. 5. Flow completion times under different scheduling schemes.

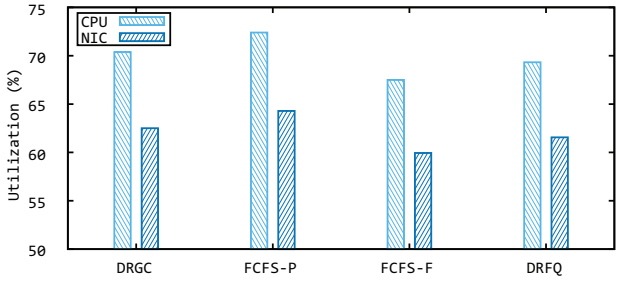


Fig. 6. Resource utilizations under different scheduling schemes.

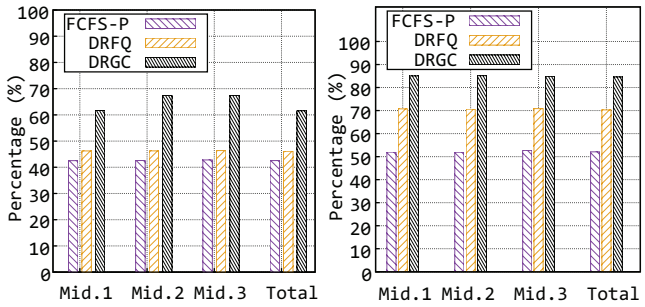
In the second scenario, flows 4, 5, and 6 belong to coflow 2. Different from the first scenario, these flows, although with different flow sizes and packet sizes, arrive at middlebox 2 sequentially. We want to evaluate the performance of DRGC in this situation. As the same in the first scenario, we predefine the completion time of coflow 2 as $40\ s$. In Fig. 4(c), flow 4 achieves its desired data rate in the first 2 seconds. Flows 5 and 6 can also get their desired data rates since they arrive at the middlebox. In this procedure, the data rate of flow 4 will not change, because it possesses the highest priority. Other flows cannot compete for resources with it. We shorten their predefined completion times to $20\ s$, such that the resources become insufficient to finish their transmissions before the predefined completion time. In Fig. 4(d), flow 4 still gets its desired data rate until completing its transmission at $20\ s$. Flow 5 can only get a part of its desired data rate since its arrival. At $4\ s$, although flow 6 has already arrived, it cannot be scheduled because the residual resources are not enough even for completing the transmission of flow 5 before $20\ s$. Thus, flow 5 monopolizes all the resources after completing the transmission of flow 4. Finally, flow 6 starts its transmission after the completion of flow 5, and finishes at $37.3\ s$.

From these two scenarios we can get the conclusion that when the resources at middleboxes are sufficient, DRGC satisfies the data rate requirements of flows such that their transmissions can be completed simultaneously at their predefined completion times. Otherwise, flows will be served sequentially based on their priorities. The ones with higher priorities can get their desired data rates, and others can only get a part of their desired data rates, or be scheduled later. Meanwhile, the lost data rates of the latter will be compensated when they become the ones with relatively higher priorities.

Completion Time Guarantee In the third scenario, we evaluate the performance of DRGC on guaranteeing the predefined completion times of coflows. We conduct our experiment with multiple middleboxes. Coflows 1 and 2 converge at middlebox 3 after passing through middleboxes 1 and 2, respectively. We suppose all the flows in these two coflows arrive at middlebox 3 simultaneously at $0\ s$, and their completion times are predefined as $50\ s$. We make such setting to maintain the middlebox at a work-conservation status, i.e., at least one resource is fully utilized at any time. As verified in the previous two scenarios, coflows 1 and 2 can safely pass through middleboxes 1 and 2 when setting their completion times as $40\ s$. The situation will not change in this scenario where these two coflows require lower data rates at middleboxes 1 and 2, respectively. We implement FCFS at the packet level and the flow level, denoted as FCFS-P and FCFS-F, respectively. In detail, FCFS-P simply schedules packets just according to packet arrival times. However, FCFS-F sequentially schedules flows one after another according to the flow arrival times. DRGC makes scheduling decisions at middleboxes, but Aalo [20] and CODA [35] are agnostic to the packet processing procedure at middleboxes. Comparing DRGC with them is extremely unfair, so we did not make related comparisons. DRFQ and MR^3 all take DRF as the scheduling criterion. MR^3 uses the round robin algorithm to reduce the scheduling overhead of DRFQ. However, they all strive to provide fair service for the passing flows. Typically, we select DRFQ to make comparisons with DRGC.

In Fig. 5, flows 1, 2, and 3 finish their transmissions simultaneously at $50\ s$, i.e., their predefined completion times, by using DRGC. Flows 4, 5, and 6 miss their predefined completion times and finish sequentially, according to their priorities. Under DRGC, coflows are scheduled based on their indicators. As aforementioned, the benefit of the coflows with higher priorities will be preferentially guaranteed when the hardware resources are insufficient. In this situation, the data rate requirements of coflow 1 will be firstly satisfied. Thus, the three flows in coflow 1 take up necessary resources so as to transmit their data, and safely complete their transmissions before their predefined completion times. Meanwhile, the residual resources are insufficient to support the transmission of coflow 2 in this procedure. Thus, flows 4, 5, and 6 will be sequentially scheduled according to their serial numbers. It is clear that DRGC takes coflows as the scheduling unit and can effectively guarantee the predefined completion times of coflows through packet-level scheduling.

With the same data rate setting under DRGC, packets of each flow will be evenly separated because all the flows have the same arrival time and completion time. FCFS-P schedules packets just according to their arrival times, and will not distinguish coflows at the packet level. Consequently, FCFS-P results in long FCTs for all the flows and these flows all miss their predefined completion times. FCFS-F achieves relatively short FCTs because we did not make data rate control under it. Flows are scheduled one after another under FCFS-F, such that they sequentially complete their transmissions. However, before the completion of the previous flow, the next flow has to wait and suffers from very long blocking time. This is unacceptable for some delay sensitive cluster applications. As a typical fairness-driven scheduling scheme, DRFQ provides



(a) Situation 1 (b) Situation 2

Fig. 7. Traffic load of valid coflows.

fair service for the passing flows. The packets of flows 1 and 4 consume more processing time on the CPU. Flow 1 completes its transmission after flow 4, because of its bigger size. Flows 2, 3, and 5 are set with the same flow size and are all backlogged on the NIC. Such that they can complete their transmissions simultaneously. However, it is clear that coflows 1 and 2 all miss their predefined completion times because of the long completion times of some individual flows under DRFQ. Thus, DRFQ is also pernicious to the benefit of coflows without coflow identification.

Resource Utilization With the same setting as in the third scenario, we evaluate the resource utilizations of the CPU and the NIC in the time dimension, i.e., the ratio of the resource usage time to the total completion time. In Fig. 6, when flows are scheduled sequentially under FCFS-F, the utilizations of the CPU and the NIC are relatively low. However, DRGC, FCFS-P, and DRFQ, which schedule flows in a mixed manner, achieve higher resource utilizations on all the resources. This observation stems from the fact that flows undergoing different network functions usually exhibit diverse preference on hardware resources. Thus, at least one resource cannot be sufficiently utilized if flows are exclusively scheduled one after another. If the packets of different flows are scheduled alternately, the complementary resource demands of flows will improve the resource utilizations. Although coflow 1 successfully finishes its transmission before its predefined completion time only by using DRGC and FCFS-F in Fig. 5, but in the long term, FCFS-F is inapplicable in the multi-resource environment.

C. Large-scale Simulation

In the previous subsection, we have verified that DRGC can detect distinct coflow traffic and sequentially guarantee their predefined completion time. Next, we use a large-scale Hive/MapReduce trace to estimate the performance of DRGC. The one-hour trace data was collected from 3000 machines within 150 racks [8][20]. Next, we will analyze some important characteristics of the trace data. The size of a coflow is defined as the sum of the sizes of all its individual flows. In the used trace data, the sizes of 96.6% coflows are smaller than 100 GB, but the residual 3.4% coflows contribute 96.6% of the total workload. Obviously, the sizes of coflows follow the heavy-tailed distribution. As aforementioned, this makes it possible to reduce the scheduling overhead of DRGC through providing transmission controls just for the coflows with huge sizes. A small part of network resources can be reserved to serve other coflows. The length of a coflow is defined as the maximal flow size among all its individual flows. The lengths of 92.6% coflows are smaller than 1 GB, but a small part of coflows contain the flows with sizes about 100 GB or 1 TB. We also measure the width of coflows, i.e., the number of parallel flows in the same coflow. The width of 79.1% coflows is smaller than 100. However, a non-ignorable part of coflows contains more than 10000 flows. In the trace data, a new coflow arrives in every $0.1\sim20$ s in most situations.

In this part of our experiments, we randomly separate the aforementioned 150 racks into three sets, each of which contains 50 racks. Each set of racks connects to one middlebox, and the individual flows randomly undergo one of the four aforementioned network functions at their directly connected middleboxes. In this way, individual flows have been almost equally distributed at three middleboxes. We set the bandwidth of the NIC as 200 Mbps. Meanwhile, the sizes of coflows, including the sizes of the individual flows, in the original trace have been scaled down by 1024 times so as to match our experiment setting. The completion times of the coflows with sizes smaller than 1 MB are set as 1 s, and the completion time of the coflow with the maximal size is set as 500 s. We also predefine that the completion times of other coflows linearly depend on their sizes. Thus, all the coflows have the chance to finish their transmissions before their predefined completion times. As aforementioned, the majority of the traffic load is contributed by a small number of coflows. The sizes of huge coflows are extremely bigger that the sizes of small ones. Striving to maximize the number of the coflows meeting their predefined completion times, which is not the scheduling objective of DRGC, results in inefficiency of the network. DRGC does not discriminate any coflows. It schedules coflows according to their arrival times. Thus, DRGC provides better support for huge coflows, whose transmissions will continue for a long time. In this part, we use the traffic load of valid coflows, i.e., the ones that successfully complete their transmissions before the predefined completion times, to estimate the performance of different schemes.

As illustrated in Fig. 7(a), the traffic load of valid coflows accounts for roughly 42% and 46% of the total workload at each middlebox by using FCFS-P and DRFQ, respectively. DRGC performs better at each middlebox, and supports additional more than 15% of the traffic load of valid coflows in total, compared with FCFS-P and DRFQ. Obviously, network resources are insufficient to satisfy all the transmission requirements of coflows in this situation. Thus, we extend the completion times of all the coflows, such that their data rates will reduce. In Fig. 7(b), when we set the completion time of the coflow with the maximal size as 1000 s, all of these schemes support more traffic load than that in the previous situation. However, DRGC still outperforms FCFS-P and DRFQ by roughly 32% and 14% in total, respectively.

In summary, DRGC achieves better performance than other schemes for the following reasons. Firstly, by using the coflow indicator, distinct coflow traffic can be recognized at the packet level under DRGC. Bear in mind that the transmission of any

individual flow in a coflow may delay the completion of the whole coflow. DRGC schedules each coflow as a whole and strives to synchronize the transmissions of its individual flows so as to protect the benefit of the coflow. However, FCFS-P and DRFQ treat flows as independent ones, thus the transmissions of the flows in the same coflow are out of step. Secondly, when the network resources are insufficient, DRGC sequentially schedules coflows according to their priorities. It centralizes resources to support the transmissions of the coflows with higher priorities. Only after that would the residual resources be utilized to serve other coflows. That means, the benefit of coflows will be sequentially guaranteed under DRGC. FCFS-P and DRFQ strive to satisfy the transmission requirements of all the coflows at all time. As a result, most coflows will miss their predefined completion times.

VI. CONCLUSION

Flow scheduling has always been a meaningful problem in data centers. Novel schemes need to be designed to satisfy the QoS requirements of flows in new scheduling scenarios. With the emergence of the concept of coflow, the benefit of coflows, rather than that of the individual flows, should be guaranteed. This problem has already been a difficult challenge in traditional networks, it becomes more complex in the multi-resource environment, where coflows encounter different transmission delays at diverse middleboxes.

In this paper, we propose DRGC to guarantee the predefined completion times of coflows in the multi-resource environment. DRGC integrates end-points and middleboxes to make fine-grained data rate control for coflows. Coflows with higher priorities are more likely to achieve their desired data rates at middleboxes such that their completion times can be sufficiently guaranteed. Meanwhile, they will not monopolize all the resources at middleboxes, and other coflows can also be scheduled by using the residual resources. We evaluated the performance of DRGC through extensive simulations and trace-driven experiments. DRGC efficiently guaranteed the predefined completion times of coflows, improved the resource utilizations and supported more workload, in comparison with other scheduling schemes.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Euro. Conf. on Computer Systems (EuroSys)*, 2007, pp. 59–72.

[3] M. Chowdhury and I. Stoica, "Coflow: a networking abstraction for cluster applications," in *Proc. of the 11th ACM Workshop on Hot Topics in Networks*, 2012.

[4] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 98–109, 2011.

[5] Z. Guo, X. Fan, R. Chen, J. Zhang, H. Zhou, S. McDirmid, C. Liu, W. Lin, J. Zhou, and L. Zhou, "Spotting code optimizations in data-parallel pipelines through periscope." in *OSDI*, 2012, pp. 121–133.

[6] J. Zhang, H. Zhou, R. Chen, X. Fan, Z. Guo, H. Lin, J. Y. Li, W. Lin, J. Zhou, and L. Zhou, "Optimizing data shuffling in data-parallel computation by understanding user-defined functions." in *NSDI*, 2012.

[7] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. of ACM SIGCOMM*, 2014.

[8] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. of ACM SIGCOMM*, 2014.

[9] Y. Zhao, K. Chen, W. Bai, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. of IEEE INFOCOM*, 2015.

[10] X. Li and C. Qian, "Low-complexity multi-resource packet scheduling for network function virtualization," in *Proc. of IEEE INFOCOM*, 2015.

[11] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. of USENIX NSDI*, 2012.

[12] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. of ACM SIGCOMM*, 2012.

[13] J. Sherry and S. Ratnasamy, "A servey of enterprise middlebox deployments," in *Technical report, EECS, UC Berkeley*, 2012.

[14] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, "G-hadoop: Mapreduce across distributed data centers for data-intensive computing," *Future Generation Computer Systems*, vol. 29, no. 3, p. 739750, 2013.

[15] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Predicting the resource consumption of network intrusion detection systems," in *Recent Advances in Intrusion Detection*. Springer, 2008, pp. 135–154.

[16] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *Proc. of ACM SIGCOMM*, 2012.

[17] W. Wang, B. li, and B. Liang, "Multi-resource round robin: a low complexity packet scheduler with dominant resource fairness," in *Proc. of IEEE ICNP*, 2013.

[18] W. Wang, B. Liang, and B. Li, "Low complexity multi-resource fair queueing with bounded delay," in *Proc. of IEEE INFOCOM*, 2014.

[19] S. Luo, H. Yu, Y. Zhao, B. Wu, and S. Wang, "Minimizing average coflow completion time with decentralized scheduling," in *Proc. of IEEE International Conference on Communications (ICC)*, 2015.

[20] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 393–406.

[21] X. S. Huang, X. S. Sun, , and T. S. E. Ng, "Sunflow: Efficient optical circuit scheduling for coflows," in *CoNEXT*, 2016.

[22] J. Nagle, "On packet switches with infinite storage," *IEEE Trans. On Communications*, vol. 35, no. 4, pp. 435–438, 1987.

[23] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 4, pp. 1–12, 1989.

[24] P. Goyal, H. M. Vin, and H. Chen, "Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 4, pp. 157–168, 1996.

[25] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control: the single node case," *ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, 1993.

[26] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, I. Stoica, and S. Shenker, "Dominant resource fairness: Fair allocation of multiple resource types," in *NSDI*, 2011.

[27] W. Wang, B. Liang, and B. Li, "Multi-resource generalized processor sharing for packet processing," in *Proc. of ACM/IEEE IWQoS*, 2013.

[28] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, "Altruistic scheduling in multi-resource clusters," in *OSDI*, 2016.
[29] J. Zhang, H. Qi, D. Guo, K. Li, W. Li, and Y. Jin, "A fair and efficient packet scheduling method in multi-resource environments," in *IEEE Transactions on Network and Service Management*, vol. 12, no. 4, pp. 605–617, 2015.
[30] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proc. of ACM SIGCOMM*, 2010.
[31] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. of ACM SIGCOMM*, 2012.
[32] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu, "Hadoopwatch: A first step towards comprehensive traffic forecasting in cloud computing," in *Proc. of IEEE INFOCOM*, 2014.
[33] M. Zaharia, M. Chowdhury, A. D. T. Das, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012.
[34] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in *Nsdi*, 2010.
[35] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *Proc. of ACM SIGCOMM*, 2016.

Jianhui Zhang received the B.E. degree from the School of Computer Science and Technology, Harbin Engineering University, China, in 2009. He received the Ph.D. degree from the School of Computer Science and Technology, Dalian University of Technology, China, in 2018. He is currently a Lecturer in the School of Information and Security Engineering, Zhongnan University of Economics and Law, Wuhan, China. His research interests include datacenter networks, network protocols and cloud computing.

Deke Guo (SM'17) received the B.S. degree in industry engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a Professor with the College of System Engineering, National University of Defense Technology, and is also with the College of Intelligence and Computing, Tianjin University. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a senior member of the IEEE and a member of the ACM.

Keqiu Li (SM'14) received the bachelor's and master's degrees from the Department of Applied Mathematics at the Dalian University of Technology in 1994 and 1997, respectively. He received the Ph.D. degree from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology in 2005. He also has two-year postdoctoral experience in the University of Tokyo, Japan. He is currently a Professor in the School of Computer Science and Technology, Dalian University of Technology, China. His research interests include internet technology, data center networks, cloud computing and wireless networks. He is a senior member of IEEE and a member of ACM.

Heng Qi is a Lecturer at the School of Computer Science and Technology, Dalian University of Technology, China. He got bachelor's degree from Hunan University in 2004. He got master's degree and doctorate degree from Dalian University of Technology in 2006 and 2012, respectively. His research interests include computer network, multimedia computing, and mobile cloud computing. He has published more than 20 technical papers in international journals and conferences, including ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMCCAP) and Pattern Recognition (PR).

Xiaoyi Tao received the B.E. degree from the School of Software Engineering, Dalian University of Technology, China, in 2011. Currently, she is a Ph.D. candidate in the School of Computer Science and Technology, Dalian University of Technology, China. Her research interests include datacenter networks, SDN networks and cloud computing.

Yingwei Jin is a Professor at the School of Management, Dalian University of Technology, China. He got his doctoral degree from Dalian University of Science and Technology in 2005. His research interests include computer network and security, Internet technology, and artificial intelligence. He has published more than 30 technical papers in international journals and conferences.