2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

OSTB: Optimizing Fairness and Efficiency for Coflow Scheduling without Prior Knowledge

Handong Qu*, Renhai Xu*, Wenxin Li†, Wenyu Qu*, Xiaobo Zhou*
*College of Intelligence and Computing, Tianjin University, Tianjin, China.
†Hong Kong University of Science & Technology, Hong Kong, China.
Wenyu Qu is the corresponding author: wenyu.qu@tju.edu.cn.

***Abstract*—Fair and efficient coflow scheduling has witnessed an increasing wave of popularity in improving application-level network performance in data-parallel clusters. With coflow size information, previous solutions perform well in simultaneously achieving fairness and fast completion for coflows. However, as coflow size is difficult to obtain in practice, scheduling coflows without prior information attracts much attention recently. Despite the prevalence, existing information-agnostic solutions either solely focus on fairness or fast completion. In this paper, we make an attempt to achieve both fairness and fast completion when scheduling coflows without prior knowledge of coflow size. Specifically, we propose an information-agnostic coflow scheduler—*OSTB*, which strives to achieve a better isolation guarantee in the long run toward fair network sharing while improving the efficiency by reducing the average CCT (Coflow Completion Time). The key idea of *OSTB* is to intelligently give the smaller coflows more opportunities to send data, while still guaranteeing that the wider or larger coflows have available bandwidth all the time. Large-scale trace-driven simulations demonstrate that *OSTB* incurs 59% fewer delayed coflows than NC-DRF when both of them use the isolation-optimal solution (e.g, DRF) as baseline. Meanwhile, *OSTB* is $1.2\times$ faster than Aalo in terms of the average CCT.**

I. Introduction

It has been widely accepted that coflow [1–3] has become an elegant model to abstract the communication patterns in distributed data-parallel applications (e.g., MapReduce [4], Spark [5]). A coflow is a set of parallel flows to transfer the intermediate results between different computation stages of a data-parallel job, and all the parallel flows must finish before a coflow is considered to be complete. In a shared datacenter network, coflows may come from multiple competing users or applications, making it extremely important to schedule coflows efficiently and fairly.

Unfortunately, one cannot achieve fairness and efficiency at the same time because: (1) A common wisdom in achieving fairness is to provide isolation guarantees on the minimum coflow progress (e.g., DRF [6] and HUG [7]), which, however, can hurt the efficiency—average coflow completion time (CCT). (2) Minimizing the average CCT only [8] turns out to be unaware of the service isolation.

Instead of enforcing strict guarantee on minimum coflow progress, Utopia [9] advocates a relaxed fairness term—*long-term isolation guarantee*: as long as a coflow completes no later than an isolation-optimal scheduler (e.g., DRF [6] or HUG [7]), its isolation is guaranteed in a long run. In such a case, Utopia can have a chance to resolve the dilemma between fairness and efficiency; even so, it requires complete prior knowledge of coflow information, e.g., coflow size. In fact, as revealed in many previous studies [10–12], the coflow-level information cannot be easily obtained in many practical cases, due to the pipelined computation, multi-wave execution and task failures. Hence, Utopia becomes inapplicable in practical scenarios, and designing information-agnostic coflow schedulers is highly desired.

To the best of our knowledge, however, no existing information-agnostic coflow schedulers can achieve the objectives of fairness and efficiency simultaneously. The crux is that they either focus on optimizing fairness by providing long-term isolation guarantee at the expense of long CCTs (e.g.,NC-DRF [12]), or improving efficiency by reducing average CCT without isolation guarantee (i.e., Aalo [10]).

In this paper, we study the problem of scheduling coflows without any prior knowledge of coflow size, with the primarily objectives of achieving fairness (i.e., *long-term isolation guarantee*) and efficiency (i.e., low average CCT) simultaneously.

Despite the difficulty in achieving the best of both worlds, we observe some design space by inheriting the advantages of Utopia [9] and Aalo [10]. Utopia [9] utilizes the completion order of coflows under a fair scheme as the scheduling order, which makes coflows likely to be completed earlier than they would have had in the fair scheme, thus accounting for long-term isolation guarantee. But without prior knowledge of coflow size, the completion order cannot be calculated. Fortunately, we found that for light-tailed distributions of coflow sizes, FIFO can also complete more coflows no later than their CCTs under DRF (see Sec.III-A). This motivates us to use FIFO for determining the scheduling orders of light-tailed coflows, and leave the remaining orders determined by existing information-agnostic coflow scheduler Aalo [10]. In order to remove the assumption of knowing coflow size, Aalo adopts the thought of D-CLAS (Discretized Coflow-aware Least-attained Service) to gradually demote coflows from higher priority queues to lower ones over time for reducing average CCT.

The above design space looks promising, but it has some limitations. First, the actual coflow sizes in each queue are not strictly subject to light-tailed distributions all the time. In many practical cases, the adjacent arrival coflows might have a lot of differences in their total actual sizes. Second, despite the coflow size divergence, one cannot make judgments early on large coflows and tend to allocate all free bandwidth to them before they are demoted to a lower priority queue. As a result, the small coflows arrived behind the large ones will be affected seriously and have arbitrary long CCTs. And multiple demotions also prolonged the completion time of the larger coflows. To solve this problem, we investigate that the high port occupancy rate can be used as an indication for the larger or wider coflows (see Sec.III-C).

Based on our investigation, we propose a novel information-agnostic coflow scheduler called as *OSTB*[1]. By utilizing a variant of MLFQ (Multi-level Feedback Queues), *OSTB* combines the port occupancy detection mechanism with D-CLAS heuristic to perceive the larger or wider coflows in advance and put them into the last queue. To avoid the larger or wider coflows being blocked for a long time, *OSTB* adopts a tunable multiplexing ratio to divide the bandwidth into two segments: a part of the bandwidth is allocated to the coflows in the last queue by fairness allocation algorithms, and another part is used for the coflows in other priority queues with FIFO manner. Thus, it can offer the smaller coflows more opportunities to send their data by priorities, while still guaranteeing that the wider or larger coflows have progress all the time. To evaluate *OSTB*, we have conducted large-scale simulations based on a real-world data trace collected from Facebook. Our experimental results can be summarized as follows. First, *OSTB* is much better than NC-DRF in long-term isolation guarantee. *OSTB* incurs 59% fewer delayed coflows than NC-DRF when both of them use the isolation-optimal solution (e.g, DRF) as baseline. Second, *OSTB* also outperforms Aalo and NC-DRF by $1.2\times$ and $1.3\times$ in terms of the average CCT, respectively.

II. BACKGROUND AND MOTIVATION

In this section, we describe our models for datacenter fabric and motivate the need for simultaneously achieving fairness and efficiency for coflow scheduling without prior knowledge.

A. Model

Datacenter fabric. Thanks to full-bisection bandwidth network is now available in datacenters, we abstract data center fabric as a giant switch with non-blocking internal data transmission [1, 8, 10]. In such a switch model, as illustrated in Fig.1, we focus only on its ingress and egress ports, where each ingress (egress) port corresponds to a machine uplink (downlink). We assume an $n \times n$ DC fabric connecting n machines. Each machine i has a full-duplex link shown as an uplink connecting ingress port i and a downlink connecting egress port $i+n$. Without loss of generalized, we assume that all links are of equal capacity normalized to one.

Coflow. A coflow consists of a collection of parallel flows that share a common performance goal across a group of machines

[1]Chinese idiom "kill two birds with one stone", used to describe an action that serves two purposes. We abbreviate it as one stone, two birds (OSTB).

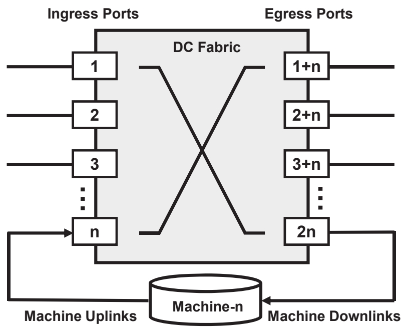


Fig. 1. An $n \times n$ non-blocking datacenter fabric with ingress/egress ports connecting to n machines.

[1]. The coflow abstraction can capture the communication requirement between two computation stages in the BSP (Bulk Synchronous Parallel) model. It builds upon the all-or-nothing property that not until all parallel flows have completed will a coflow complete.

As a common practice [7, 13], we use the vector $\vec{S}_k = \left\langle S_k^1, S_k^2, \ldots, S_k^{2n} \right\rangle$ to describe the demand of coflow-k, where S_k^i captures the amount of data transferred on link-i. In order to perceive the demand correlation of coflows on each link, we identify the most heavily loaded link among all links of coflow-k as its *bottleneck link*, and let the demand of bottleneck link as the *bottleneck demand* of coflow-k, i.e., $\bar{S}_k = \max_i S_k^i$. Then we can get the demand correlation vector $\vec{C}_k = \left\langle C_k^1, C_k^2, \ldots, C_k^{2n} \right\rangle$ of coflow-k, where $C_k^i = S_k^i / \bar{S}_k$. The correlation vector means that for every bit of data transmits on *bottleneck link*, at least C_k^i bits data should be transmitted on the link-i. If we denote the bandwidth allocated to coflow-k on link-i by A_k^i, then $P_k = \min_i \left\{ A_k^i / C_k^i \right\}$, the minimum demand-normalized bandwidth allocation over all links, captures the progress of coflow-k. Intuitively, the transmission rate on the slowest link determines the progress of the entire coflow [9], which critically affects the CCT.

B. Objectives

In this paper, we focus on optimizing two primary objectives for non-clairvoyant coflow scheduling: average CCT (efficiency) and long-term isolation guarantee (fairness).

Average CCT: Recent studies [2, 14] have shown that intermediate data transmission accounts for more than 50% of job completion time, which means the coflow completion time significantly affects the performance of cluster applications. Hence, an efficiency-oriented network scheduler should strive to minimize the average CCT.

Long-term isolation guarantee: In a shared datacenter, coflows expect guarantee on the minimum progress to ensure isolation performance among them. A common wisdom in achieving fairness is to seek an allocation that maximizes the minimum progress among all coflows, i.e.,

$$\text{maximize} \quad \min_k P_k. \tag{1}$$

Above objective just focus on instantaneous allocation for coflow scheduling, which means the progress of each coflow

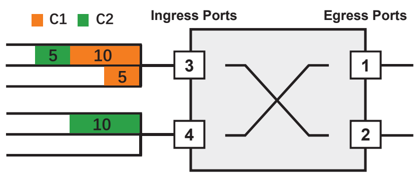
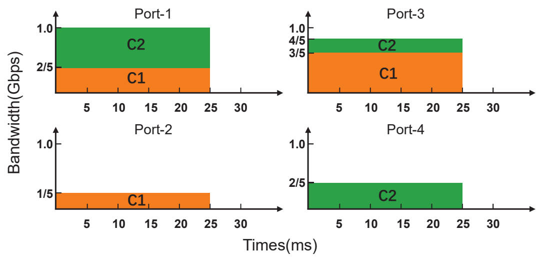
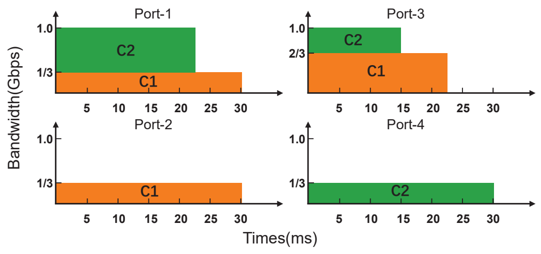


Fig. 2. Assuming two coflows $C1$ and $C2$ are scheduled on four ingress/egress 1Gbps ports: $C1$ with demand vector $\vec{S}_{c1} = \langle 10, 5, 15, 0\rangle$Mb and $C2$ with demand vector $\vec{S}_{c2} = \langle 15, 0, 5, 10\rangle$Mb on each port.



(a) DRF



(b) NC-DRF

Fig. 3. (a) and (b) show the bandwidth allocation under DRF and NC-DRF, respectively. Illustrating that NC-DRF is still far from DRF in isolation guarantee, and the inefficiency of NC-DRF on minimizing average CCT.

must be maintained at the attainable maximum level P^* in each instant, however, it can hurt the efficiency. To solve this problem, many recent studies [9, 12] advocate a relaxed fairness term—*long-term isolation guarantee*: as long as a coflow completes no later than an isolation-optimal scheduler (e.g., DRF [6] or HUG [7]), its isolation is guaranteed in a long run. Hence, a fairness-oriented scheduler should guarantee: (1) The proportion of the coflows that complete no later than they are under an isolation-optimal scheduler. (2) The worst isolation performance, which guarantees coflows cannot be arbitrarily blocked for a long time.

C. Motivation

For information-agnostic coflow scheduling, Aalo and NC-DRF are two representative schedulers for efficiency and fairness, respectively.

Aalo. To minimize the average CCT without prior knowledge, Aalo [10] first proposes D-CLAS policy, which decreases coflow priority only when the number of bytes it has sent exceeds some predefined thresholds. Overall, it adopts FIFO manner for coflows in each queue and smallest-coflow-first for coflows among queues. Aalo does help to reduce the average CCT, but it ignores the isolation guarantee among contending coflows, which brings some coflows with longer CCTs. Actually, it has been shown in [12] that Aalo can dramatically speed the coflow completion, however, provides no isolation guarantee, resulting some coflows delayed more than $100\times$ compared with their CCTs under DRF.

NC-DRF. Unlike Aalo, recently proposed NC-DRF [12] only aims to optimize isolation guarantee. Without coflow size information, it utilized DRF algorithm by attaining estimated coflow demand correlation vectors according to the flow count information on each link. NC-DRF holds that the disparity of flow sizes within a coflow is usually small owing to the load balancing principle. However, NC-DRF is not good for efficiency like most isolation guarantee algorithms [6, 7], which always ignore the actual demand and blindly force a same progress to all coflows. Besides, NC-DRF also performs poorly in long-term isolation guarantee even if the disparity of flow sizes within a same coflow is small, which has been shown in NC-DRF's evaluations that coflows are delayed by 68% on average as compared with the DRF.

Next, we use a toy example to illustrate this point. Suppose there are two coflows in the network contending on four 1Gbps ports, as shown in Fig.2. Especially, we set the disparity of flow sizes is small. Under DRF scheme, Coflow $C1$ has a correlation vector $\vec{C}_{c1} = \left\langle \frac{2}{3}, \frac{1}{3}, 1, 0\right\rangle$ and coflow $C2$ has $\vec{C}_{c2} = \left\langle 1, 0, \frac{1}{3}, \frac{2}{3}\right\rangle$. It increases the progress of each coflow to the maximum level $P^* = 1/\max_i \sum_k C_k^i = \frac{3}{5}$, where i and k represent each port and coflow in the fabric. The bandwidth allocation of the two coflows as shown in Fig.3(a), and the CCTs of two coflows both are $25ms$. If we use the NC-DRF scheme, the correlation vector will be $\vec{C}_{c1} = \left\langle \frac{1}{2}, \frac{1}{2}, 1, 0\right\rangle$ and $\vec{C}_{c2} = \left\langle 1, 0, \frac{1}{2}, \frac{1}{2}\right\rangle$ for $C1$ and $C2$, respectively. Theoretically, the allocated progress is $P^{*'} = \frac{2}{3}$ as shown in Fig.3(b). But the actual progress drops to $\frac{1}{2}$ according to coflow actual demands. Moreover, the CCTs of two coflows both are $30ms$ and increased by 20% on average compared with DRF. Hence, there is still a gap between NC-DRF and DRF in terms of both instantaneous and long-term isolation guarantee.

To summarize, neither Aalo nor NC-DRF can perform well on fairness and fast completion at the same time. Motivated by this, we think that achieving these two objectives simultaneously through non-clairvoyant coflow scheduling is an important and urgent research topic.

III. CHALLENGES AND SOLUTIONS

In this section, we present our key insight of achieving both fast completion and long-term isolation guarantee without prior knowledge of coflow size. We also analyze the challenges to naively implement this insight and give a reasonable solution.

A. Key Insight

Drawing on Utopia [9], the priority scheduling method can achieve the best of both fairness and efficiency. It utilizes the completion order of coflows under a fair scheme (i.e., DRF [6] or HUG [7]) as the scheduling order, which makes coflows likely to be completed earlier than they would have had in the fair scheme, guaranteeing fast completion at the same time. However, for the non-clairvoyant coflow scheduling, the completion order cannot be attained.

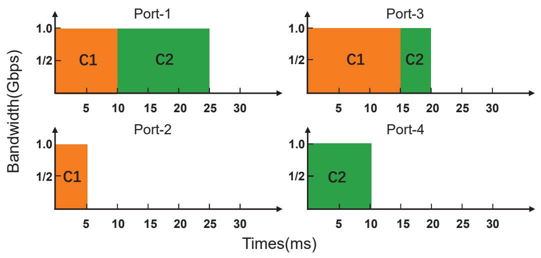


Fig. 4. The bandwidth allocation under FIFO for two coflows $C1$ and $C2$ in Fig.2.

To address this problem, we switch our thinking to other available priority policies for information-agnostic scheduling, such as FIFO scheme can provide optimal efficiency for light-tailed distributions of coflow sizes [15]. Hence, we consider that FIFO may also perform well on the long-term isolation guarantee for light-tailed coflows. Let's look back at the two coflows $C1$ and $C2$ in Fig.2. They are same in total coflow size. Fig.4 shows that FIFO scheduler would have resulted in a lower average CCT ($20ms$) than NC-DRF and DRF if $C1$ was scheduled before $C2$, and it would have been the same if $C2$ was scheduled before $C1$. The reason is that two coflows with similar size will finish roughly at the same time by adopting DRF or NC-DRF—both taking twice as much time as a single coflow. It worth noting that FIFO can speed up $C1$ ($C2$) without delaying the latter $C2$ ($C1$) when compared with DRF. Hence, FIFO can be utilized to achieve our two objectives simultaneously for light-tailed coflows.

B. Challenges

Someone may think that Aalo happens to be a non-clairvoyant scheduler that can provide FIFO policy, but why Aalo is still very poor in long-term isolation guarantee, isn't it a contradiction? We analyzed the root causes from two aspects: (1) In each queue, the actual sizes of coflows are not strictly subject to light-tailed distributions all the time. (2) Among queues, a larger coflow may suffer from longer CCT due to its lower priority and multiple demotions.

As we know that coflow sizes are subject to heavy-tailed distributions in most datacenters, and the adjacent arrival coflows might have a lot of differences in their total sizes. However, Aalo cannot make judgments earlier on the larger coflows, and tend to allocate all free bandwidth to them before they are demoted to a lower priority queue according to D-CLAS policy. Thus, there inevitably exists a situation that a large coflow will prior to a small coflow be scheduled by using FIFO in each queue, which makes some small coflows arrived behind the larger ones will be penalized for the longer waiting time. And multiple demotions can also severely harm the efficiency of the larger coflows. Hence, the key to benefit both small and large coflows is to seek a solution that can perceive the larger coflows as early as possible. Next, we show how to achieve it.

C. Solution

Definition 1 (Port occupancy rate): Let N_k be the number of ports occupied by coflow-k , and N be the number of the ports in the entire datacenter fabric (i.e., $N = 2n$ in Fig.1). We define the coflow *port occupancy rate* R_k equals N_k divided by N, i.e.,

$$R_k = \frac{N_k}{N}. \tag{2}$$

Even though the flow size is often unavailable, the flow on which link (port) of a coflow can be easily obtained through the machine learning techniques [11]. Accordingly, the port occupancy rate for each coflow can be computed.

We calculated the correlations between coflow sizes and port occupancy rates of a real-world data trace with 526 coflows collected from Facebook [16]. According to the statistics, the coflow sizes are almost no smaller than 1000MB when the port occupancy rates are more than 0.5. Conversely, 87% of coflows are smaller than 1000MB and 75% of coflows are smaller than 100MB when the port occupancy rates are smaller than 0.5. Hence, it is reasonable to say that in most practical cases the high port occupancy rate can be used as an indication for the larger coflows under the information-agnostic situation.

Given this observation, we still utilize the total number of bytes sent by all flows of a coflow to determine the priorities of coflows among MLFQ. Moreover, we perceive the coflows with high port occupancy rates in advance, then put them into the last queue and allocate bandwidth by adopting fairness algorithm with limited multiplexing. Finally, we use FIFO for determining the scheduling orders of light-tailed coflows in each queue. The effectiveness of this approach as follows.

- First, the coflow's port occupancy rate is high means its size is large with great possibilities, and most of them will be eventually demoted to the last queue. Even if the coflow is not a large one in size, it must be wide and occupy bandwidths on lots of ports. Correspondingly, there are more coflows arrived behind it have no bandwidth available and wait to be scheduled on the same queue. Hence, If we can perceive such coflows with high port occupancy rates as early as possible, more small coflows arrived behind the large ones will get chance to transfer their data timely.
- Second, the remaining smaller coflows that account for a large proportion in the number of coflows have a few differences in size, basically satisfying the light-tailed distribution in each queue. Using FIFO manner to schedule the remaining coflows is not only good for their efficiency, but also can provide better long-term isolation guarantee (see Sec.III-A).
- Finally, with limited multiplexing, the larger coflows can avoid being blocked for a long time and get available bandwidth all the time. Moreover, the time losses caused by multiple demotions will be also reduced greatly.

IV. *OSTB*

In this section, we present the detailed design of *OSTB*, an algorithm that can provide better long-term isolation guarantee while achieving efficiency improvement compared with existing state-of-the-art non-clairvoyant coflow schedulers [10, 12].

A. Architecture

In general, *OSTB* builds upon MLFQ, and relies on the port occupancy detection mechanism and D-CLAS heuristics. In order to avoid full decentralization causes coflow scheduling pointless and full centralization introduces high overheads for small coflows, *OSTB* also adopts loosely-coordinated and implements global and local control. Fig.5 shows the architecture of *OSTB*. Next, we will introduce it from following two parts.

Center Coordinator. The center coordinator plays an important role in the overall scheduling, which is used for long-term global coordination. Just like previous work [10], it collects the information of coflows from each daemon running on each end host every $O(10)$ milliseconds, then determines the global coflows ordering according to the D-CLAS. Finally, it will send out the updated schedule order and globally observed coflow information to all the local daemons for transmission. Moreover, *OSTB* makes some changes to it.

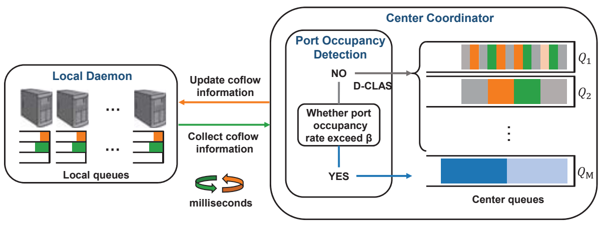


Fig. 5. Architecture of *OSTB*

The main differences of center coordinator between *OSTB* and Aalo lie in that *OSTB* uses the last queue as a fair queue where the coflows with high port occupancy rates are scheduled by adopting fairness allocation algorithms with limited multiplexing. Besides, *OSTB* adds the port occupancy detection mechanism. When the coordinator collects information of a new coflow that arrives after previous synchronization, *OSTB* first determines the coflow should be put into which queue according to the port occupancy rate whether exceeds a predefined threshold β. Note in *OSTB*, the order of coflows in the last queue does not change according to D-CLAS, unless there is a coflow is demoted from Q_{M-1} to the last queue or a new coflow have high port occupancy rate exceeds β.

Local Daemon. Each end host will run a local daemon to monitor the runtime status of coflows and report the information to the center coordinator. As shown in Fig.5, the local daemon is responsible for short-term local prioritization. Each local daemon schedule local coflows using the last-known global ordering. When coflows arrive between two consecutive synchronization, it will be put into the first queue in their local priority queues temporarily in the short term, and rearrange them with global schedule as soon as updated

Algorithm 1 Online coflow scheduling

```
1:  procedure CoordinatorDetection(Coflow $C_k$)
2:      if $C_k$ is a new coflow then
3:          Count up the number of ports occupied by coflow $C_k$, i.e, $N_k$
4:          if $N_k/N > \beta$ then
5:              $Q_M \leftarrow Q_M \cup \{C_k\}$                ▷ The rate is high
6:          else
7:              $Q_m \leftarrow Q_m \cup \{C_k\}$        ▷ According to D-CLAS
8:                                                        ▷ $m \in [1, M-1]$
9:          end if
10:     else if $C_k$ is an old coflow then
11:         Update the coflow priority according to D-CLAS among $Q_m$   ▷ $m \in [1, M-1]$
12:     end if
13: end procedure
14:
15: procedure CoflowComplete(Coflow $C_k$)
16:     $Q_m \leftarrow Q_m \setminus \{C_k\}$                   ▷ $m \in [1, M]$
17: end procedure
```

global information arrives. For the flows with same source-destination, the latter flow belongs to another coflow will replace the previous one that has just completed for work conservation.

B. Online Coflow Scheduling

The *OSTB* coordinator utilizes the port occupancy detection mechanism combined with D-CLAS to divide coflows into different multi-level priority queues. In this paper, we assume there are a small number of M queues in total, from the highest priority queue Q_1 to the lowest priority queue Q_{M-1}, and the last queue Q_M is a fair queue as shown in Fig.5. We employ exponentially-spaced scheme to set the queue thresholds. Specifically, we define T_i is the threshold that a coflow is demoted from Q_i to Q_{i+1}, where $i \in [1, M-1]$. Our scheduler is an efficient online scheduler for dynamic coflow arrivals. In general, a coflow may experience the following three types of events during its lifetime.

1) **Arrival:** When a coflow arrives before a certain global coordination, it will be enqueued into the local highest priority queue for short-term scheduling. And once the next global coordination begins, the new coflow is either placed in the last queue due to high port occupancy rate or placed in other priority queues according to D-CLAS policy (lines 4-8 of Alg.1).
2) **Activities:** Coflows whose information has been collected by the global coordinator can be divided into two types. The first kind of coflow is those that will be demoted to Q_{i+1} from Q_i as its total number of bytes have sent exceed queue threshold T_i (line 11 of Alg.1). And if a coflow exceed the threshold of queue Q_{M-1}, even such a situation is rare, it can be put into the last queue normally. Conversely, the second type of coflow refers to those with high port occupancy rates. Once placed in the last queue, they are no longer affected by

their total number of bytes have sent. And they will be scheduled in a fair manner by limited multiplexing until their completion.

3) **Completion:** Once a coflow completes, the cluster application de-registers it from *OSTB*, and it will be removed from its current queue in the center coordinator immediately (line 16 of Alg.1).

C. Bandwidth Allocation

OSTB is designed to optimizing both on fairness and fast completion. Given scheduling scheme above, our algorithm allocate bandwidth as shown in Alg.2.

Formally, let Q represent the collection of coflow queues. Let f_k^{ij} represent an individual flow in coflow C_k transferring data from uplink-i to downlink-j in the fabric, and the r_k^{ij} be the bandwidth allocated to f_k^{ij}. Besides, We conceptually divide the bandwidth B_i on each port into two segments: L_i is allocated to the smaller coflows in each Q_m, $m \in [1, M-1]$ by FIFO, while F_i is the reserved bandwidth used for transmission of the minority larger coflows in the last queue Q_M by adopting non-clairvoyant fairness algorithms. And they all maintain the current remaining bandwidth during algorithm execution process. The ratio α between the two segments is predefined by the scheduler (i.e., $F_i = \alpha B_i$), and can be adjusted dynamically based on network status. Next, we will show the details of our allocation algorithm as follows.

Algorithm 2 Bandwidth allocation algorithm

```
1:  procedure ALLOCBANDWIDTH(Queues $Q$)
2:      Initialize remaining bandwidth $B_i \leftarrow 1$ on all link-$i$
3:      CoordinatorDetection($C_k$) for all coflow-k in coordinator in this epoch
4:      for all $C_k \in Q_M$ do
5:          Allocation bandwidth for $C_k$ using fairness allocation algorithm, e.g., NC-DRF
6:          $F_i \leftarrow F_i - r_k^{ij}$
7:          $F_j \leftarrow F_j - r_k^{ij}$
8:      end for
9:      for all link-$i$ do
10:         $L_i \leftarrow L_i + F_i$            ▷ Ensure work-conservation
11:     end for
12:     for $m = 1$ to $M-1$ do
13:         for all coflows $C_k \in Q_m$ do
14:             for all flows $f_k^{ij} \in C_k$ do
15:                 $r_k^{ij} \leftarrow$ Max-min fair share
16:                 $L_i \leftarrow L_i - r_k^{ij}$
17:                 $L_j \leftarrow L_j - r_k^{ij}$
18:             end for
19:         end for
20:     end for
21:     Distribute unused bandwidth to all coflows $C \in Q_m$, $m \in [1, M]$    ▷ Ensure work-conservation
22: end procedure
```

Fairness Manner: We start with detecting all coflows by using *CoordinatorDetection*(C_k) in Alg.1, then allocate bandwidth for the coflows in Q_M by non-clairvoyant fairness algorithm (lines 4-8 of Alg.2). And we do not restrict *OSTB* to any specific fairness algorithms such as NC-DRF [12], PS-P [17] or per-flow fairness [18] strategies. The key is the larger coflows in the last queue can always get progress guarantee and avoid suffering from long completion time.

Priority Manner: To make full use of the idle bandwidth, we first add the remaining bandwidth F_i to L_i on each port (line 10 of Alg.2). Next, *OSTB* utilizes strict priority queueing for allocation across multi-level feedback queues and adopting FIFO in each queue. For the intra-coflow scheduling, it allocates bandwidth for every flows by using max-min fair policy, and updates the available bandwidth on each link (lines 15-17 of Alg.2). Note that L_i here also represents the remaining bandwidth of B_i.

Work Conserving: After processes above, there may be remaining bandwidths in the fabric that can be utilized by some coflows in the last queue. To better utilize resource and improve efficiency, we traverse all links with available bandwidth and check whether there are any flows that can be sent (line 21 of Alg.2). Subject to the capacity constraints in the coupled links, the bandwidth added to a flow should be the smaller available bandwidth between its uplink and downlink.

V. Evaluation

In this section, we compare *OSTB* against three representative allocation algorithms—Aalo [10], NC-DRF [12] and Utopia [9]—using trace-driven simulations. As explained Aalo and NC-DRF are the state-of-the-art information-agnostic schedulers for efficiency and fairness respectively, and Utopia is the optimal scheduler for simultaneously achieving the two objectives with prior knowledge.

A. Workload and Setup

Workload. To emulate realistic scenarios, we use the one-hour workload trace with 526 coflows in Coflow-Benchmark [16], which is based on a Hive/MapReduce trace collected by Chowdhury et al. from a 3000-machine, 150-rack Facebook cluster. The benchmark is scaled down to a 150-port fabric, where all mappers (reducers) in the same rack are combined into one rack-level mapper (reducer), as production clusters are oversubscribed in core-rack links and simulating rack-level is sufficient for them [18].

To better understand the impact on different coflows, we categorize coflows into four bins based on their lengths and widths. In general, we consider a coflow is short (long) if the size of its longest flow is less (greater) than 5MB, and narrow (wide) if the number of its flows is less (greater) than 50. the detailed distribution results are shown in Table I.

TABLE I
Coflows binned by their lengths (Short and Long) and widths (Narrow and Wide) in the Coflow-Benchmark[16].

Coflow Bin	**SN**	**LN**	**SW**	**LW**
% of Coflows	60%	16%	12%	12%
% of Bytes	0.01%	0.11%	0.88%	99.00%

Setup. In our simulation, we modeled the fabric as a 150×150 non-blocking switch with 150 ingress (egress) ports corresponding to the uplinks (downlinks) of 150 racks connected to it. And we set the bandwidth of each port to 1Gbps. For the Multi-level feedback queue, we employed the same size thresholds as those in Aalo. That is, the threshold to demoted coflows is $Q_i = Q_1 \times 10^i$ where $i \in [1, M-1]$ and $Q_1 = 10$MB. The number of queues is set to 10.

For the last queue, we schedule the larger or wider coflows with NC-DRF fairness algorithm. Besides, we set the port occupancy rate threshold β to 0.7, which means if the number of ports occupied by a coflow exceeds $300 \times 0.7 = 220$, the coflow will be put into the fair queue. In general, the greater occupancy rate threshold β, the smaller multiplexing ratio α. Because if β is greater, there will have fewer coflows be put into the last queue. Correspondingly, the multiplexing bandwidths allocated to coflows in the last queue should be reduced. In this paper, we set the multiplexing factor α to 0.3, which balances β and α, making them complementary.

B. Evaluation Metrics

Long-term isolation guarantee: For evaluate the long-term isolation guarantee of different schedulers, we utilized *Normalized CCT* [12] as its metric, which is defined, for each coflow, as the CCT under the compared scheduler normalized by that under clairvoyant isolation-optimal scheduler DRF, i.e.,

$$\textit{Normalized CCT} = \frac{\text{Compared CCT}}{\text{CCT under DRF}}$$

In general, the *normalized CCT* of a coflow less than one means the coflow's isolation is guaranteed in a long run. As mentioned earlier (see Sec.II-B), to achieve long-term isolation guarantee better, we should satisfy two optimization objectives as follows:

- First, making the *normalized CCT* of more coflows less than one. It ensures more coflows get long-term isolation guarantees.
- Second, the value of *normalized CCT* should have an upper limit as small as possible. It guarantees the worst isolation performance for coflows that cannot attain long-term isolation guarantees.

Efficiency: We use the *Improvement multiple* to evaluate the efficiency improvement of *OSTB* compared to other schedulers. and define it as the average CCT of a scheduler normalized by *OSTB*'s average CCT, i.e.,

$$\textit{Improvement multiple} = \frac{\text{Compared average CCT}}{\textit{OSTB}\text{'s average CCT}}$$

If the *improvement multiple* of a scheduler is greater (smaller) than one, *OSTB* is faster (slower). Hence, this metric can measure how much faster *OSTB* than other schedulers in efficiency.

C. Improvements

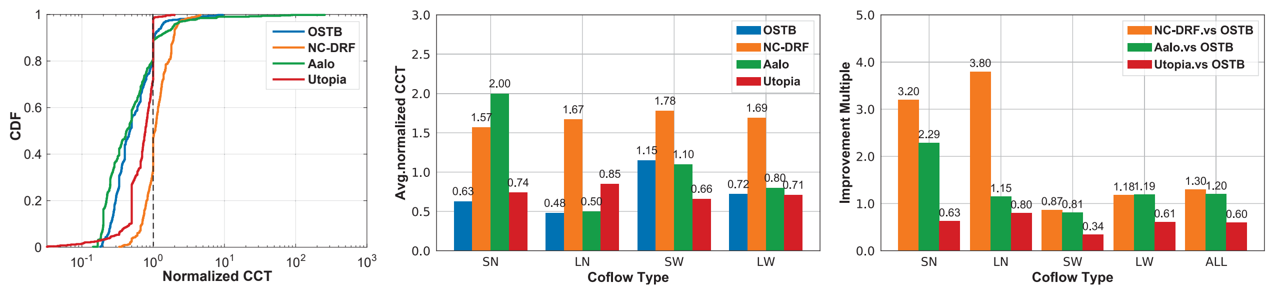
Fairness improvement: We plotted the distributions of *normalized CCT* for each coflow under different schedulers, as shown in Fig.6(a). And Table.II presents the detailed statistical information. We can observe that *OSTB* incurs 59% fewer delayed coflows than NC-DRF when both of them use the isolation-optimal solution (i.e., DRF) as baseline. Specifically, *OSTB* can make the proportion of delayed coflows whose *normalized CCTs* larger than one dramatically reduced to 11.2% from 70% under NC-DRF. Interestingly, Aalo can also ensure long-term isolation guarantees for as many coflows as *OSTB*. But the maximum *normalized CCT* in Aalo is much larger than 100, while *OSTB* has little difference with NC-DRF, none of them are larger than 10. That means there will be some coflows blocked arbitrarily for a long time in Aalo, which seriously affects the fairness among contending coflows. Besides, *OSTB* consistently outperforms NC-DRF under the metrics of minimum, average and 95th percentile *normalized CCT*. Even compared with Utopia, *OSTB* still have a smaller average *normalized CCT*. This fully demonstrates the superiority of *OSTB* in long-term isolation guarantee.

In Fig.6(b), we measured the average *normalized CCTs* in four different bins. It worth noting that, in SN and LN bins, *OSTB* has a smaller average *normalized CCT* compared with all other schedulers. This is because the benefit we receive from giving the smaller coflows more opportunities to send their data by FIFO manner. However, for the coflows in SW bin, *OSTB* is slightly worse than Aalo and Utopia. We attribute this to that some coflows in SW bin will be put into the last queue due to high port occupancy rates. Even they are short in length, we cannot perceive them timely due to the limitation of port occupancy detection mechanism. Hence, the coflows with both characteristics of short and wide may be affected slightly. As for the average *normalized CCT* in LW bins, *OSTB* is close to the optimal solution Utopia. This illustrates that *OSTB* also can offer the larger coflows with long-term isolation guarantees on average by using limited multiplexing.

TABLE II
STATISTICAL SUMMARY OF NORMALIZED CCTS FOR DIFFERENT SCHEDULERS.

	Min	**Mean**	**95th**	**Max**	**Perc(Nor.CCT>1)**
OSTB	0.18	0.68	1.31	9.54	11.2%
NC-DRF	0.37	1.68	2.15	4.71	70.0%
Aalo	0.14	1.51	1.92	256	10.5%
Utopia	0.03	0.75	1.00	2.00	2.0%

Efficiency improvement: For better illustrating the efficiency improvement, we contrast *OSTB* with Aalo and NC-DRF. Fig.6(c) shows that *OSTB* reduces average CCT by up to $1.2\times$ and $1.3\times$ in comparison to Aalo and NC-DRF, respectively. Similar to the effect in fairness, *OSTB* performs worst for the coflows in SW bin. Across all bins, we observe more improvements in SN and LN bins over SW and LW bins, meaning that narrow coflows will gain more speed up than wide ones in *OSTB*. As expected, *OSTB*'s average CCT have little difference with Aalo except in SN bin—a strong evidence that *OSTB* can avoid long waiting time for the smaller coflows



(a) Distribution of normalized CCT (b) Average normalized CCT (c) Improvement multiple

Fig. 6. Illustrates that *OSTB* can perform better than existing non-clairvoyant schedulers both on fairness and fast completion. (a) Distributions of *normalized CCTs* under different schedulers. (b) Average *normalized CCTs* in different coflow bins. (c) *OSTB*'s improvement compared against three different schedulers.

without affecting efficiency of the larger ones. The proportion of coflows in SN bin is larger, therefore speeding up their completion time will also increase the overall efficiency. To understand how far we are from the optimal solutions, we have compared *OSTB* against Utopia. Fig.6(c) presents that the *Improvement multiple* in all bins are smaller than one, which illustrates that *OSTB* still worse than the optimal efficiency due to lacking of prior knowledge of coflow size.

VI. Conclusions

In this paper, we focused on the non-clairvoyant coflow scheduling problem with the objectives of optimizing fairness and efficiency simultaneously. Existing non-clairvoyant works either focus only on improving efficiency by reducing average CCT (e.g., Aalo), or optimizing fairness by providing long- term isolation guarantee at the expense of long CCTs (e.g., NC-DRF). Fortunately, we found that for the light-tailed coflows, FIFO not only can perform well in minimizing average CCT, but also can provide more coflows with long-term isolation guarantee under the information-agnostic situation. Hence, combined with port occupancy detection mechanism, we craft a heuristic algorithm *OSTB* to handle this problem effectively. Extensive simulations based on realistic workloads indicate that *OSTB* outperforms existing non-clairvoyant approaches on both objectives. The theoretical analysis of the optimal multiplexing ratio and port occupancy rate threshold for *OSTB* is considered as our future work.

Acknowledgment

This work is supported in part by the National Key Research and Development Program of China No. 2018YFB1004703, in part by the State Key Program of National Natural Science of China under Grant No. 61432002, in part by the National Natural Science Foundation of China - Guangdong Joint Funds under Grant No. U1701263, in part by the the NSFC-General Technology Basic Research Joint Funds under Grant No. U1836214, in part by the National Natural Science Foundation of China under Grant No. 61702365, 61672379 and 61772112, in part by the Natural Science Foundation of Tianjin under Grant No. 17JCQNJC00700, 17ZXRGGX00150 and 18ZXZNGX00040.

References

[1] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. of ACM Workshop on Hot Topics in Networks*, 2012.
[2] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proc. of ACM SIGCOMM*, Toronto, Canada, 2011.
[3] W. Li, X. Yuan, K. Li, H. Qi, and X. Zhou, "Leveraging endpoint flexibility when scheduling coflows across geo-distributed datacenters," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 873–881.
[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[5] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. of Usenix HotCloud*, 2010.
[6] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Nsdi*, 2011.
[7] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "Hug: Multi-resource fairness for correlated and elastic demands." in *NSDI*, 2016.
[8] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. of ACM SIGCOMM*, Chicago, IL, USA, 2014.
[9] L. Wang, W. Wang, and B. Li, "Utopia: Near-optimal coflow scheduling with isolation guarantee," 2018.
[10] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. of ACM SIGCOMM*, 2015.
[11] 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.
[12] L. Wang and W. Wang, "Fair coflow scheduling without prior knowledge," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 22–32.
[13] W. Wang and A.-L. Jin, "Friends or foes: Revisiting strategy-proofness in cloud network sharing," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*, 2016.
[14] W. Li, D. Guo, A. X. Liu, K. Li, H. Qi, S. Guo, A. Munir, and X. Tao, "Coman: managing bandwidth across computing frameworks in multiplexed datacenters," *IEEE transactions on parallel and distributed systems*, vol. 29, no. 5, pp. 1013–1029, 2017.
[15] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. of ACM SIGCOMM*, 2014.
[16] "Synthesized data from real-world traces of data-intensive applications for coflow benchmarking." [Online]. Available: https://github.com/coflow/coflow-benchmark
[17] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," *ACM SIGCOMM Computer Communication Review*, 2012.
[18] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "Eyeq: Practical network performance isolation at the edge," *REM*, vol. 1005, no. A1, p. A2, 2013.