Fair Coflow Scheduling without Prior Knowledge

Luping Wang, Wei Wang
Hong Kong University of Science and Technology
{lwangbm, weiwa}@cse.ust.hk

***Abstract*—Coflow scheduling improves the networking performance at the application level in datacenters. Ideally, a coflow scheduler should provide tenants with *isolation guarantees* to achieve predictable networking performance. Existing works in this regard (e.g., DRF [1] and HUG [2]) are limited to the *clairvoyant* scheduling, in that the complete knowledge of coflow sizes is assumed to be available before the communication starts. However, this assumption does not hold for many applications with pipelined computation, in which clairvoyant coflow schedulers become *inapplicable*.**

To bridge this gap, we develop a new non-clairvoyant coflow scheduler, called *Non-Clairvoyant DRF* (NC-DRF), which provides isolation guarantees between contending coflows *without prior knowledge* of coflow size. We show that NC-DRF achieves provable isolation guarantees *in the long run*. Cluster deployment and trace-driven simulations show that with NC-DRF, coflows are only delayed by 68% on average as compared with the clairvoyant, isolation-optimal DRF [1]. NC-DRF also outperforms existing alternatives (e.g., per-link fairness [3]) by $1.7\times$ in terms of the average coflow completion time.

I. Introduction

Communications in data-parallel applications typically involve a collection of parallel flows between groups of machines (e.g., shuffle phase in a MapReduce job)—known as *coflows* [4]. The coflow abstraction exposes the application-level semantics to the network and captures the *all-or-nothing* communication requirement of data-parallel jobs: a coflow is not considered complete until *all* its constituent flows have completed. As the network bandwidth in the datacenter is increasingly contended by coflows of different tenants and applications, it is critical for a network scheduler to schedule these coflows in a *fair* and *efficient* manner to improve the application-level networking performance.

Achieving *optimal isolation guarantees* between contending coflows arises as the top requirement in shared cloud environments [3], [5]–[9]. Ideally, a scheduler should provide each coflow with the isolation guarantees on the minimum bandwidth allocation, so as to achieve predictable *coflow completion time* (CCT).

Many coflow schedulers have been proposed in this regard by means of *fair network sharing*. The state-of-the-art solutions include the recently proposed HUG [2] and its variant [9], under which coflows expect the *optimal* isolation guarantees. Central to these schedulers is to enforce the DRF-like (Dominant Resource Fairness [1]) bandwidth allocation among coflows, which critically requires the *prior knowledge* of *coflow sizes* before the data transfer begins. In this sense, existing fair coflow schedulers are all *clairvoyant*.

Unfortunately, prior work [10] shows that in many data-parallel applications, it is *impossible* to obtain the coflow knowledge *a priori*, mainly due to the pipelined computation, the wave-based job executions and the presence of task failures. Consequently, clairvoyant coflow schedulers, such as HUG [2], remain *inapplicable* in many practical scenarios. To our knowledge, the development of a *non-clairvoyant* coflow scheduler towards the optimal isolation guarantees has so far received little attention in the literature.

However, achieving isolation guarantees without prior knowledge of coflows can be *challenging*. In the non-clairvoyant setting, the conventional definition of isolation guarantees (i.e, the minimum progress across all coflows [2]) is no longer feasible. In fact, the coflow *progress*, defined as the minimum demand-normalized allocation across links, cannot be computed without knowing the data transfer demand (i.e., coflow size). On the other hand, existing non-clairvoyant network schedulers are unable to optimize the networking performance at the application level. For example, network schedulers based on per-flow fairness (TCP) provides *no isolation* for data-parallel applications [3]; tenant-level schedulers based on per-link fairness [3], [11] fall short as they are agnostic to coflow demand correlations and can only provide *suboptimal* isolation guarantee [2].

In this paper, we aim at addressing these challenges while making the following two contributions.

First, unlike existing clairvoyant coflow schedulers [2], [9], we for the first time put forward the definition of isolation guarantee under non-clairvoyant coflow scheduling. Specifically, we use the fair scheduler with complete coflow information (e.g., DRF [1] or HUG [2]) as a baseline. We say that a coflow scheduler provides optimal isolation guarantee if each coflow is guaranteed *no longer* CCT than it would otherwise have had under the baseline scheme. Our definition is based on the fact that the effect of isolation guarantee can only be observed by an application when the coflow completes.

Second, we propose a new non-clairvoyant coflow scheduler, called Non-Clairvoyant DRF (NC-DRF), which *provides isolation guarantees between contending coflows without prior knowledge of coflow size*. Our key insight is to use the *flow count* on each link in a coflow as an indication of the data transfer demand, based on which we employ the isolation-optimal Dominant Resource Fairness (DRF) [1] algorithm for bandwidth allocation between coflows. We find this approach is effective for two reasons. First, even though the flow size is often unavailable, the flow count on each link in a coflow can be easily obtained through either the scheduler API [10] or the

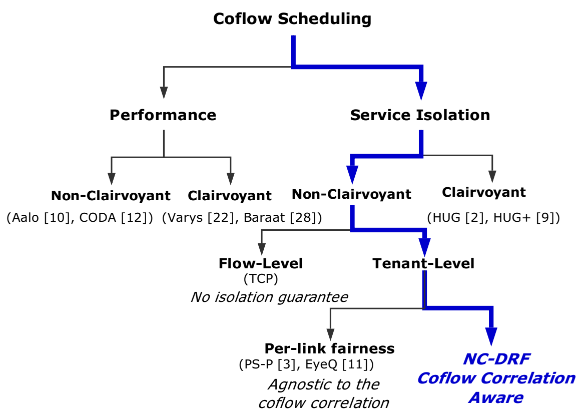


Fig. 1: Design space for coflow scheduling.

machine learning techniques [12]. Second, according to the load balancing principle followed by most of the data-parallel applications [13]–[15], the *disparity* of flow sizes within a coflow is usually small. Meaning, the flow count on each link has a *strong correlation* to the amount of data transferred on that link, and can hence be used to identify the coflow demand correlation across multiple links. This provides us an opportunity to employ the isolation-optimal scheduling algorithms (e.g., DRF [1]) through non-clairvoyant coflow scheduling.

We show that NC-DRF provides *provable* long-term isolation guarantee under some practical assumptions. We evaluated our proposed NC-DRF through a real-world deployment on a 60-machine Amazon EC2 [16] cluster, as well as in simulations over production traces collected from a 3000-machine cluster at Facebook [17]. Our evaluation results confirm that NC-DRF provides long-term fairness with guaranteed isolation between contending coflows. In particular, with NC-DRF, coflows are only delayed by 68% on average as compared with the isolation-optimal DRF [1]. NC-DRF also outperforms PS-P [3] by $1.7\times$ in terms of the average CCT.

Fig. 1 surveys the design space for coflow schedulers (details in Sec. II-B), in which our proposed NC-DRF is placed in context by the thick lines. In particular, NC-DRF strives to isolate coflow completions without prior knowledge about the coflow size, while still being aware of the coflow demand correlations.

II. Model and Background

In this section, we describe our models for both datacenter fabric and coflow scheduling problem. We also survey the background information of existing works and motivate the need for a new non-clairvoyant coflow scheduling policy striving to isolate coflow completions.

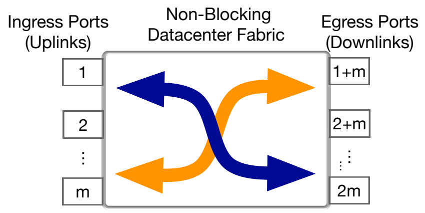


Fig. 2: An $m \times m$ non-blocking datacenter fabric with ingress/egress ports connecting to m machines.

A. System Model

Datacenter network model. Thanks to the recent efforts in datacenter (DC) fabrics [18]–[20], full bisection bandwidth network is now available in production datacenters [21]. This allows us to model the DC network as one *non-blocking switch* where the edges—machine uplinks and downlinks—are the only sources of contention [4], [22]. Fig. 2 illustrates the $m \times m$ non-blocking DC fabric connecting m machines through full-duplex links, where link-i and link-$(i+m)$ correspond to the uplink and downlink of machine-i, respectively. Without loss of generality, we assume that all links are of equal capacity normalized to one.

Coflow abstraction. A coflow includes a collection of parallel flows transferring data between two computation stages in the BSP (Bulk-Synchronous Parallel) model [4], which is used to convey the application-level communication requirement. Central to coflow scheduling is the *all-or-nothing* communication requirement of data-parallel jobs: not until all constituent flows have completed will a coflow complete.

Formally, we characterize coflow-k by its *demand vector* $\mathbf{d}_k = \langle d_k^1, \ldots, d_k^{2m} \rangle$, where d_k^i denotes the amount of data transferred on link-i. In order to capture the most heavily loaded one among all the $2m$ links, we identify the link with the largest transferred data size as the *bottleneck link*, i.e., $b_k = \arg\max_i d_k^i$, and let $\bar{d}_k = \max_i d_k^i$ be the *bottleneck demand*. In addition, to better characterize the demand correlation across links, we define *correlation vector* $\mathbf{c}_k = \langle c_k^1, \ldots, c_k^{2m} \rangle$, where c_k^i is the total amount of data transferred on link-i normalized by the bottleneck demand, i.e., $c_k^i = d_k^i / \bar{d}_k$. Meaning, for every bit coflow-k transfers on the bottleneck link, *at least* c_k^i bits should be transferred on link-i.

Given demand and correlation vectors (i.e., $\mathbf{d}_k$ and $\mathbf{c}_k$), the network scheduler determines, for each coflow-k, the bandwidth allocation $\mathbf{a}_k = \langle a_k^1, \ldots, a_k^{2m} \rangle$, where a_k^i is the share of bandwidth on link-i. Once the allocation has been given, the coflow transmission progress is bottlenecked on the *slowest* link. Formally, we measure the *progress* of a coflow as the minimum demand-normalized allocation across links, i.e.,

$$P_k = \min_{i: c_k^i > 0} a_k^i / c_k^i. \tag{1}$$

Intuitively, a coflow progress captures the attainable transmission rate on the slowest link, which critically determines the coflow completion time (CCT).

B. Prior Art

Non-Clairvoyant Coflows Scheduling. Even though the amount of data each flow transfers can be known *a priori* in some data-parallel applications such as MapReduce, prior work [10] reveals that fully characterizing the coflow sizes in advance is *infeasible* in many cases.

First, multi-stage applications such as Apache Tez [23] and MapReduce Online [13] allow data to be pipelined between subsequent computation stages and transferred as soon as it is generated. In this case, it is hard to know the coflow size before the communication completes. Second, the wave-based execution [24] leads to variance in the start time of different flows within a coflow, and the count of flows changes over time. Finally, the presence of task failures and straggler mitigation techniques [15], [25], [26] result in redundant flows, making it hard to know the exact count of flows or their endpoints *a priori*.

Given these observations, clairvoyant schedulers assuming complete prior knowledge of coflows remain *inapplicable* in a large count of application scenarios.

Prior art of coflow scheduling. As shown in Fig. 1, coflow scheduling has been extensively studied in recent years, among which the existing works could be broadly categorized into two approaches, depending on the objectives of the scheduling policy: *performance-optimal heuristics* and *isolation-optimal fair network sharing*. Both approaches could be further divided into two categories: *clairvoyant* coflow scheduling, which assumes complete prior information (e.g., the count of flows, their sizes and endpoints) of coflow, and *non-clairvoyant* coflow scheduling, which assumes no prior coflow knowledge.

1) Performance-optimal heuristics: To speed up job completion, network operators need to finish as many coflows as possible, each in its fastest possible way. A performance-optimal scheduler should therefore strive to minimize the average coflow completion time (CCT).

- *Clairvoyant schedulers:* The state-of-art coflow scheduler settling for minimizing the average CCT goes to Varys [22], which is arguably of the best performance. Varys employs the *Smallest-Effective-Bottleneck-First* heuristic, which generalizes the Shortest-Job-First (SJF) heuristic to the context of coflow scheduling to prioritize small coflows. Besides, Orchestra [27] and Baraat [28] use FIFO-based scheduling to reduce the average CCT. The former is a centralized scheduler, while the latter is decentralized and avoids head-of-line blocking by means of fair sharing.
- *Non-clairvoyant schedulers:* Aalo [10] improves its clairvoyant predecessor Varys in that the prior knowledge about the coflow size is no longer needed. To minimize the average CCT without knowing the coflow sizes, Aalo proposes Discretized Coflow-Aware Least-Attained Service (D-CLAS). D-CLAS divides coflows into multiple priority queues and schedules them in the FIFO order within each queue. Aalo achieves comparable performance to Varys. While Aalo needs to modify data-parallel applications to identify coflows, the recently proposed CODA [12] takes a step forward to automatically identify coflows without modifying applications.

2) Isolation-optimal fair network sharing: On the other hand, many schedulers turn to fair network sharing to provide optimal isolation between coflows. In a shared datacenter, coflows expect guarantees on the minimum bandwidth to achieve predictable performance. Specifically, given an allocation, the *isolation guarantee* is defined as the minimum progress across all coflows, i.e., $\min_k P_k$, where P_k is given by Eq. (1). A coflow scheduler optimizes the isolation guarantee if the minimum progress is maximized.

- *Clairvoyant schedulers:* Assuming complete coflow knowledge, fair schedulers [1], [2], [9] allocate each coflow a fair share of network bandwidth in the datacenter fabric, hence isolating the completion of each coflow from another. The state-of-the-art fair schedulers include the recently proposed HUG [2] and its variant [9], under which coflows expect the *optimal* isolation guarantee. Specifically, HUG [2] employs a two-stage bandwidth allocation algorithm. In the first stage, it enforces a DRF [1] allocation to achieve the maximum isolation guarantee, i.e., $\text{maximize} \min_k P_k$. In the second stage, it strives to attain the highest-possible network utilization by allocating spare bandwidth under the constraint that no coflow is allocated more bandwidth in a link than its progress.
- *Non-clairvoyant schedulers:* Unfortunately, the isolation-optimal algorithms like DRF [1] and HUG [2] are *inapplicable* for non-clairvoyant coflow scheduling: without the knowledge of coflow size, the coflow progress cannot be computed. In fact, the design of non-clairvoyant coflow scheduler towards optimal isolation guarantees has so far received little attention in the literature. To our knowledge, without the flow size knowledge, flow-level fairness (TCP) and per-link fairness [3], [11] are the only two non-clairvoyant policies for fair network sharing. However, both approaches fall short: the former fails to isolate coflow completions [3]; the latter can only achieve sub-optimal isolation guarantee due to its unawareness of the coflow demand correlation.

To summarize, clairvoyant coflow scheduling [2], [9], [22], [28] has been well studied in the literature, including both performance-optimal heuristics and isolation-optimal fair network sharing. However, achieving isolation guarantees through non-clairvoyant coflow scheduling remains an open research topic.

III. Objective and Challenge

In this section, we first justify that for a coflow, the number of flows on each link can be obtained *a priori*. We then clarify the objective of our proposed non-clairvoyant coflow scheduler. Finally, we discuss the non-trivial challenges to achieve this objective.

A. Objective

Availability of flow counts. Given the recent advances in non-clairvoyant coflow scheduling [10], [12], we note that even though the coflow size is often unavailable beforehand [10], the count of flows on each link can still be obtained *a priori*.

In particular, for each single flow, the information of which coflow it belongs to and its endpoints could be *directly obtained* if the data-parallel applications implement the specially designed *API* provided by coflow schedulers, as shown in the work of Aalo [10]. Alternatively, this knowledge could be measured by the *automatically identifying techniques*, such the clustering-based machine learning algorithm proposed by CODA [12].

Scheduling objective. Given this observation, in this paper, our objective is to *design a new non-clairvoyant coflow scheduler without prior knowledge of coflow sizes (i.e, the amount of data transferred in each flow), while still providing isolation between contending coflows in a shared network*. In other words, the only information we assume for each coflow is the count of flows transferred on each link.

B. Challenges

1) Definition of the isolation guarantee: The first challenge is how should we quantify the isolation guarantee provided by a non-clairvoyant coflow scheduler.

It is well known that in clairvoyant coflow scheduling, we measure the attainable transmission rate of a coflow by its *progress*, based on which the isolation guarantee is correspondingly defined as the minimum progress across all coflows, i.e., $\min_k P_k$.

Given this definition, the optimal isolation guarantee in clairvoyant coflow scheduling is usually achieved by means of *fair network sharing* [1], [2]. In particular, the fair coflow scheduler *equally* increases the progress of all the coflows to the *maximum level*, i.e.,

$$P^* = \frac{1}{\max_i \sum_k c_k^i}. \tag{2}$$

Unfortunately, this definition of isolation guarantee is infeasible in non-clairvoyant coflow scheduling. Specifically, for each coflow-k, without the prior knowledge of coflow size, the normalized demand on link-i (i.e., c_k^i) will not be revealed until the coflow's completion. Therefore, the instantaneous coflow progress can not be measured. Meaning, it is *impossible* to seek the isolation guarantee by enforcing the equal *instantaneous* coflow progress in non-clairvoyant coflow scheduling.

To address this challenge, we for the first time put forward a new definition of the isolation guarantee in non-clairvoyant coflow scheduling.

We note that from the perspective of applications or tenants, the effect of isolation guarantee can only be observed in the *long run* upon the coflow *competition*, since the users only care about the completion of the coflows rather than the instantaneous bandwidth sharing. Given this observation, if we use the fair scheduling (e.g., DRF [1] or HUG [2]) as a *baseline* algorithm, from an application's view, as long as its coflow completes *no later* than it would have had in the baseline scheme, its isolation is guaranteed in the long run. This motivates us to seek a *long-term* isolation guarantee with the following definition.

Definition 1 (Long-term isolation guarantee): For coflow-k, let F_k be its CCT under scheduler S, and F_k^{D} be the CCT in a fair scheme that enforces *instantaneous* fair allocation with the optimal isolation guarantee P^*, e.g., DRF [1]. We say scheduler S provides *long-term isolation guarantee* if it completes each coflow-k with CCT *no longer* than F_k^{D} multiplied by a *constant* ratio R, i.e.,

$$F_k \leq R F_k^{\mathrm{D}}. \tag{3}$$

As the long-term isolation guarantee above is defined based on the coflow completion time instead of the coflow progress, it is naturally suitable for the non-clairvoyant coflow scheduling. Our objective in this paper is to achieve the long-term isolation guarantee with a relatively small constant ratio.

2) Hardness in capturing coflow-level communication patterns: Given the definition presented in (3), the second challenge is how to achieve such isolation guarantee. Particularly, without the knowledge of coflow size, how to bound the CCT of each coflow with a constant ratio compared with the clairvoyant fair schedulers (DRF or HUG).

To our knowledge, when assuming *no* prior information of coflow size, flow-level fairness and tenant-level per-link fairness [3] are the only two choices in order to isolate network services. However, both these policies fail in efficiently leveraging the *coflow-level communication pattern*.

Prior work [3] shows that traditional flow-level network scheduling policies, such as per-flow fairness (TCP), source-destination pairs, and sources alone, are unaware of the coflow abstraction, providing *no* application-level isolation guarantee. Making things worse, under TCP, a tenant could take an *arbitrarily* high share of network bandwidth by initiating more flows or creating more communication endpoints, which will inevitably drag down the share of its contenders.

Tenant-level policy which is aware of the abstraction of coflow turns to be a better alternative that can avoid the problem above. In particular, FairCloud's Proportional Sharing on Proximate Links (PS-P) policy [3] based on per-link fairness achieves the service isolation at coflow-level by *equally* dividing the bandwidth among tenants. Unfortunately, PS-P captures *no* coflow-level communication pattern, i.e., coflow *demand correlation*, providing only *suboptimal* isolation guarantees [2]. A coflow has *correlated* bandwidth demand across multiple links, and a coflow scheduler must deal with multi-resource scheduling with *coupled* constraints on uplinks/downlinks. Therefore, schedulers ignoring the inherent coflow demand correlation will inevitability delay the coflow completions.

Example. To illustrate this point, we refer to a simple example as shown in Fig. 3, where two coflows contend on four 1-Gbps links. In particular, coflow-A consists of two flows, transferring 100 Mb data from link-1 and link-2 to link-4, respectively, hence has demand $\mathbf{d}_A = \langle 100, 100, 0, 200 \rangle$; similarly, coflow-$B$ has $\mathbf{d}_B = \langle 0, 200, 100, 100 \rangle$. We illustrate the network

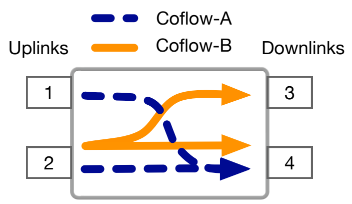


Fig. 3: Illustration of the suboptimal isolation guarantee of per-link fairness. Two coflows contend on four 1-Gbps links: coflow-A with demand vector $\mathbf{d}_A = \langle 100, 100, 0, 200\rangle$ and coflow-B with $\mathbf{d}_B = \langle 0, 200, 100, 100\rangle$.

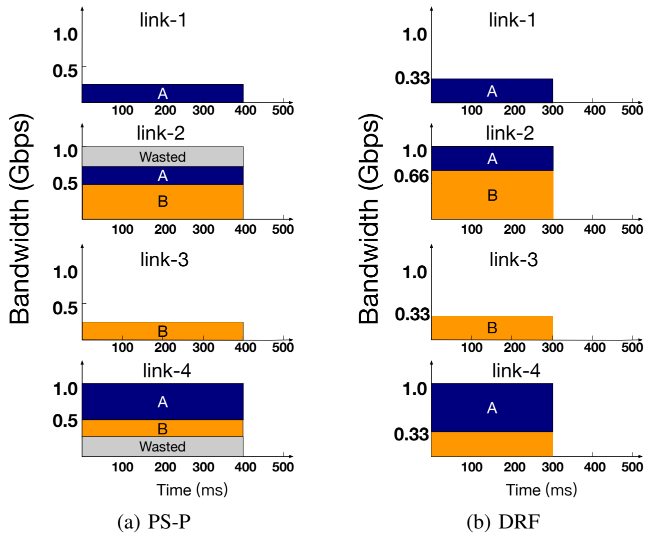


Fig. 4: Illustration of the network bandwidth sharing and coflow CCTs for the example shown in Fig. 3. (a) Bandwidth share under PS-P. (b) Bandwidth share under DRF.

bandwidth sharing and coflow CCTs under both PS-P and DRF in this example, as shown in Fig. 4.

Under PS-P policy, both coflows will *equally* share link-2 and link-4 with the same bandwidth of 0.5 Gbps. However, for coflow-A, due to the agnosticism to coflow demand correlation, PS-P can only by default choose to allocate its obtained 0.5 Gbps on link-4 *evenly* to the two flows from link-1 and link-2, each with transferring rate of 0.25 Gbps, respectively, wasting 0.25 Gbps obtained on link-2. As a *posterior knowledge*, link-2 is actually the *bottleneck link* of coflow-B, and the wasted 0.25 Gbps would have sped its completion. Similarly, coflow-B wastes 0.25 Gbps bandwidth on link-4, which would have sped the completion of coflow-A, as shown in Fig. 4a. Such bandwidth wasting will inevitably delay the coflow completions as opposed to the DRF scheme depicted in Fig. 4b.

Unnecessary bandwidth wasting in PS-P. We attribute the root cause of this bandwidth wasting to PS-P ignoring the coflow-level communication pattern, especially the *coupled constraints* between each uplink/downlink and its corresponding downlinks/uplinks. Specifically, after the inter-coflow scheduling stage where the scheduler evenly divides the bandwidth on each link among coflows, a coflow's obtained bandwidth, though allocated, may not be fully utilized due to the flow *conservation constraint* [29], i.e., the total amount of uplink/downlink bandwidth must match that of its coupled downlinks/uplinks. In other words, more bandwidth allocation, either on uplink or downlink, will be wasted.

To summarize, achieving isolation guarantee without prior knowledge of coflow size is challenging, while existing alternatives are problematic. This motivates us to seek a new fair scheduling policy that is able to efficiently leverage the coflow-level communication pattern.

IV. Non-Clairvoyant DRF

In this section, we first illustrate our intuition to reveal the coflow demand correlation, which is to use the flow count on each link in a coflow as an important indication of the actual coflow demand. Based on this intuition, we then present a new non-clairvoyant network scheduler, called *Non-Clairvoyant DRF (NC-DRF)*, achieving isolation guarantee between contending coflows. We further show that our proposed NC-DRF provides *provable* long-term isolation guarantee.

A. Key Intuition

Flow count as an indication. Our discussion above reveals that the key to seek network service isolation in the non-clairvoyant coflow scheduling is to be aware of coflow-level communication pattern. We argue the *flow count* on each link in a coflow can be used to efficiently describe the coflow demand correlation across multiple links.

Intuitively, when the amount of data to be transferred in in each single flow does not differ a lot from each other, it is reasonable to say that the count of flows on each uplink/downlink could *approximately represent* the demand correlation between these links.

Consider an extreme condition where a coflow consists of flows with identical flow size, i.e., all the flows in a coflow have the same amount of data to transfer. In this scenario, although we assume that the flow size is unknown by the underlying network scheduler, the isolation-optimal algorithms, such as DRF [1], can still be implemented. Specifically, DRF requires no information about the exact value of coflow demand on each link, but the *demand correlation*, i.e., $\mathbf{c}_k = \langle c_k^1, \ldots, c_k^{2m}\rangle$ of each coflow. When assuming identical flow size, the flow count on each link can definitely characterize the demand correlation, based on which the scheduler is able to enforce the isolation-optimal DRF allocation.

Flow size disparity in a coflow. While applications in practice are less likely to transfer all the data with flows of identical sizes, we argue that the *disparity* of flow sizes within a particular coflow could *not* be arbitrarily high.

In order to identify the flow size disparity within coflow C_k, we use e_k to denote the differential degree of transmission demand among all the *utilized* links, i.e.,

$$e_k = \bar{d}_k / \min_i d_k^i. \tag{4}$$

Intuitively, the larger e_k is, the higher the differential degree of transmission demand will C_k have.

Considering the efficiency in the production datacenters, most data-parallel applications, such as MapReduce [15] and streaming applications [13], [14], widely follow the *load balancing* principle. Specifically, when a data-parallel job is running on a homogeneous cluster, load balancing is critical, as the overall computation speed is gated by the slowest performer. Given this observation, it is reasonable to say that in most practical cases, applications are *less likely* to generate flows with sizes of sparse distribution. In other words, in these scenarios, the flow count on each link has strong correlation with the amount of data to be transferred, thus could be considered as an *indication* of the coflow demand correlation.

Admitting the limited disparity of coflow demand across links, the isolation-optimal scheduling policies such as DRF [1] can be implemented even without the prior knowledge of coflow size, while providing us an opportunity to convey the coflow-level communication requirement to the underlying network scheduler.

Key Insight. Following this observation, our key insight is to *enforce the Dominant Resource Fairness (DRF) allocation based on the flow count on each link in a coflow.*

Compared with prior works [3], [11], this simple approach can potentially provides following three benefits:

1) *It enables a non-clairvoyant network scheduler to employ the advanced isolation-optimal scheduling algorithms (e.g., DRF).* Unlike PS-P which naively enforces an equal bandwidth sharing on each link to all the coflows, our intuition provides a global view of communication demands inter and intra coflows, breaking the limitation of the applicability for isolation-optimal policies under non-clairvoyant coflow scheduling.
2) *It avoids the bandwidth wasting problem arises during the intra-coflow scheduling stage.* As shown in Fig. 4a, evenly dividing the bandwidth on each link to coflows will *inevitably* lead to bandwidth wasting. In contrast, employing our intuition can avoid such wasting as much as possible, as a coflow's bandwidth allocation on each link is *proportional* to the flow count on this link, ensuring the allocated bandwidth on each uplink/downlink can be more effectively utilized by its coupled downlinks/uplinks.
3) *It allocates bandwidth in each coflow with a bias towards small flows, which will potentially speed the coflow completion.* In more general cases where a coflow consists of flows with various flow sizes, a scheduler implementing our intuition will allocate flows with *small* size more bandwidth than they would have had under DRF. In other words, it imitates the Shortest-Flow-First heuristic in the performance-optimal coflow schedulers [22], hence potentially reduces the average CCTs.

B. Non-Clairvoyant DRF

Based on the insight above, we next illustrate our proposed new coflow scheduler, Non-Clairvoyant DRF (NC-DRF).

Algorithm 1 Non-Clairvoyant DRF

```
1: procedure ALLOCBANDWIDTH(Coflows $\mathbb{C}$)
2:     Initialize remaining b/w $R_i \leftarrow 1$ on all link-$i$
3:     for $k = 1$ to $|\mathbb{C}|$ do                ▷ Intra-coflow statistics
4:         for all link-$i$ do
5:             Count up flow counts on this link, i.e, $n_k^i$
6:         $\bar{n}_k \leftarrow \max_i n_k^i$
7:         for all link-$i$ do
8:             $\hat{c}_k^i \leftarrow n_k^i / \bar{n}_k$
9:     $\hat{P}^* \leftarrow 1/\max_i \sum_k \hat{c}_k^i$            ▷ Inter-coflow fair sharing
10:    for $k = 1$ to $|\mathbb{C}|$ do                ▷ Intra-coflow allocation
11:        $r_k \leftarrow \hat{P}^* / \bar{n}_k$
12:        for all flow $f_k^{ij} \in C_k$ do
13:            $r_k^{ij} \leftarrow r_k$                    ▷ Bandwidth for each flow
14:            $R_i \leftarrow R_i - r_k^{ij}$
15:            $R_j \leftarrow R_j - r_k^{ij}$
16: procedure NC-DRFOFFLINE(Coflows $\mathbb{C}$)
17:    allocBandwidth($\mathbb{C}$)
18:    Distribute unused bandwidth to all $C \in \mathbb{C}$
19: procedure NC-DRFONLINE(Coflow $C$, Bool isArrival)
20:    if isArrival then
21:        $\mathbb{C} \leftarrow \mathbb{C} \cup \{C\}$                ▷ Coflow $C$ arrives
22:    else
23:        $\mathbb{C} \leftarrow \mathbb{C} \setminus \{C\}$                ▷ Coflow $C$ completes
24:    NC-DRFOffline($\mathbb{C}$)
```

Offline Scheduling. As a starting point, we consider a scheduling problem with N coflows $C_1, C_2, \ldots, C_N$ contending the network resources in an *offline* scenario, where all coflows arrived at time 0. We therefore obtain a scheduling scheme $\mathbb{C} = (C_1, \ldots, C_N)$.

Bandwidth Allocation. Given scheduling scheme above, our algorithm allocates bandwidth based on DRF policy which employs the flow count to describe the demand correlation across multiple links. We summarize the entire allocating process as a procedure, called `allocBandwidth(ℂ)`, as shown in Algorithm 1.

Formally, let f_k^{ij} represent an individual flow in C_k transferring data from uplink-i to downlink-j in the fabric, and r_k^{ij} be the bandwidth allocation to this flow. We also maintain the amount of available (remaining) bandwidth on link-i using a parameter R_i, which is initialized to 1.

We start by capturing the demand correlation for each $C_k \in \mathbb{C}$. In particular, for each link-i, we collect the count of flows with which coflow C_k transferring data on link-i, denoted by n_i^k. After getting the flow count vector contained all the $2m$ links, i.e., $\mathbf{n}_k = \langle n_k^1, \ldots, n_k^{2m} \rangle$, we identify the *largest* flow count as the *flow-count bottleneck*, i.e., $\bar{n}_k = \max_i n_k^i$ and the corresponding link as *flow-count-bottleneck link*, i.e., $\hat{b}_k = \arg\max_i n_k^i$. Similar to the characterization of a clairvoyant coflow, we further calculate the *flow-count correlation vector* $\hat{\mathbf{c}}_k = \langle \hat{c}_k^1, \ldots, \hat{c}_k^{2m} \rangle$, where $\hat{c}_k^i = n_k^i / \bar{n}_k$.

Given the flow-count correlation above, we adopt the *DRF* algorithm in the following inter-coflow scheduling stage. Formally, we increase the equal sharing on each flow-count-bottleneck link to the maximum level, denoted by $\hat{P}^*$, i.e.,

$$\hat{P}^* = \frac{1}{\max_i \sum_k \hat{c}_k^i}. \quad (5)$$

Considering the flow count, i.e., n_k^i as an indication of the data transferring demand, i.e., d_k^i, NC-DRF achieves generalized max-min fairness in multi-resource coflow scheduling problem by enforcing such maximized equal sharing on flow-count-bottleneck links.

Based on $\hat{P}^*$ obtained by Eq. (5), we next allocate bandwidth to each coflow $C_k \in \mathbb{C}$. Specifically, on each link-i, coflow C_k is allocated bandwidth proportional to its flow count on this link, i.e., $a_k^i = \hat{c}_k^i \hat{P}^*$.

After getting a_k^i for each coflow, we step into the *intra-coflow* scheduling stage, where the scheduler has to decide, for each coflow, how to distribute the obtained bandwidth on each link-i, i.e., a_k^i, to individual flows on this link. Due to the agnosticism to the flow size, NC-DRF follows the same intro-coflow allocation method as PS-P, which is to *evenly* divides the bandwidth allocation on link-i to all the flows on this link, i.e., $r_k^{ij} = a_k^i / n_k^i$. This intra-coflow scheduling method enforces an *equal* transferring speed for all the individual flows within a coflow. This identical speed could be calculated by $r_k = \hat{P}^* / \bar{n}_k$, where r_k denotes this equal flow transferring bandwidth in coflow C_k, as shown in Algorithm 1 line 11.

For each round of allocating bandwidth to flow f_k^{ij}, our algorithm also updates the remaining bandwidth on both its uplink and downlink, which is used in the following work-conserving technique.

Retaining Work Conservation. To achieve work-conserving allocations, after the DRF-style allocation presented above, we further distribute the remaining bandwidth on each link, if any, to coflows. For simplicity, we adopt the backfilling strategy which evenly allocates the unused bandwidth on each link to all the active flows within all the coflows, subject to the capacity constraints in the coupled links, i.e., $w_k^{ij} = \min\{\frac{u^i}{\sum_k n_k^i}, \frac{u^j}{\sum_k n_k^j}\}$, where w_k^{ij} refers to the work-conserving allocation to flow f_k^{ij}, while u^i and u^j denote the unused bandwidth on the uplink and downlink of this flow, respectively.

We illustrate the entire offline scheduling in NC-DRF to a procedure `NC-DRFOffline(`$\mathbb{C}$`)`, as shown in Algorithm 1.

From offline to online. In order to handle dynamic coflow arrivals (departures) in *online* scheduling scenarios, we maintain a list of *active* coflows, denoted by $\mathbb{C}$. Upon an arrival (departure) of a coflow C, we insert (remove) it into (from) the list, followed by allocation update, summarized as a procedure `NC-DRFOnline` in Algorithm 1.

Example. To better illustrate the algorithmic behaviors of NC-DRF, we refer back to the previous example in Fig. 3, where two coflows contending on four 1-Gbps links. In this case where all the flows in a coflow has identical flow size, our proposed NC-DRF provides the same scheduling behavior as the isolation-optimal DRF policy, as shown in Fig. 4b.

Particularly, coflow-A has one flow transferring data from link-1 to link-4, and another flow from link-2 to link-4, therefore has flow count correlation vector $\hat{\mathbf{c}}_A = \langle 0.5, 0.5, 0, 1 \rangle$; coflow-$B$ has $\hat{\mathbf{c}}_B = \langle 0, 1, 0.5, 0.5 \rangle$. Therefore, the maximum equal sharing on the flow-count-bottleneck links is calculated by $\hat{P}^* = 1/\max_i \sum_k \hat{c}_k^i = 2/3$. Since the flow-count bottleneck of both coflows is 2, i.e., $\bar{n}_A = \bar{n}_B = 2$, all the four flows in this example will get transferring bandwidth of $1/3$ Gbps, speeding the completion of both coflows by 25%.

We highlight two points from this toy example:

- Even without further work-conserving allocation, in this example, our proposed NC-DRF can fully utilize the bandwidth resources on both link-2 and link-4, which is actually the bottleneck link of coflow-B and coflow-A, respectively.
- In such scenarios where a coflow consists of flows with identical flow size, our NC-DRF shows the same scheduling behavior as the isolation-optimal DRF algorithm, yet requiring no information of coflow size.

NC-DRF vs. Per-link Fairness. Given the discussion above, regardless the work-conserving allocation, PS-P leads to worse scheduling performance compared with NC-DRF, unnecessarily delaying the coflow completion. This is due to the bandwidth wasting in the intra-coflow scheduling stage.

In contrast, NC-DRF will never encounter such wasting. Actually, due to the agnosticism to the flow sizes, both PS-P and NC-DRF adopt a naive equalized allocation policy during intra-coflow scheduling stage, which evenly distributes a coflow's obtained bandwidth to all the flows on each link. However, under NC-DRF, a coflow's allocated bandwidth on each link-i is in proportion to its flow count, i.e., $a_k^i = n_k^i \hat{P}^* / \bar{n}_k$, which definitely matches the allocations on its coupled links. We attribute this to NC-DRF's awareness of coflow demand correlations.

C. Long-time Isolation Guarantee

We next present that NC-DRF provides long-term isolation guarantee defined in Sec. III-B under offline case with some reasonable assumptions.

Assumptions. Without loss of generality, let coflows be indexed in ascending order of their bottleneck demands $\bar{d}_k$, i.e.,$\bar{d}_1 \leq \bar{d}_2 \ldots \bar{d}_N$. For each coflow $C_k \in \mathbb{C}$, we identify the count of its used uplinks and downlinks as M_k and R_k, respectively. We make two assumptions in the following discussion:

First, we assume that for each coflow $C_k \in \mathbb{C}$, it uses more uplinks than downlinks, i.e., $R_k < M_k$.

Second, we assume that each downlink used in C_k has flows with identical sizes from all the M_k uplinks, i.e., $d_k^{1j} = d_k^{2j} \cdots = d_k^{M_k j}$, where d_k^{ij} denotes the flows size of f_k^{ij}. This assumption is reasonable, as most data-parallel applications follow the load balancing principle, avoiding flows within a coflow have arbitrarily high flow size disparity, as we have discussed above.

Isolation Guarantee. Eq. (4) has defined the flow size disparity within a coflow. We further denote the largest e_k among all the coflows by $e_{\max}$, i.e., $e_{\max} = \max_k e_k$. Formally, we have the following theorem:

Theorem 1 (Long-term isolation guarantee): Assume that coflows $\mathbb{C}$ arrived at time 0. For coflow $C_k \in \mathbb{C}$, let F_k be its CCT in Algorithm 1, and F_k^{D} its CCT in DRF. We bound the maximum completion delay beyond DRF as follows:

$$F_k \leq e_{\max} F_k^{\mathrm{D}}, \tag{6}$$

where $e_{\max}$ is the maximum demand disparity in a coflow.

We make two remarks on Theorem 1:

1) *The ratio bound $e_{\max}$ is a small constant in production datacenters.* As we have discussed above, Given the load balancing principle followed by most data-parallel applications, the demand disparity among multiple links in a coflow will not be arbitrarily large.
2) *The ratio bound is established as a worst-case guarantee, but coflows usually complete faster with much shorter average CCT.* In fact, as we shall show in Sec. V, with NC-DRF, coflows are only delayed by 68% on average as compared with DRF.

We next give a proof sketch of Theorem 1. The complete proof is deferred to our technical report [30] due to space constraints.

Proof Sketch. We start by considering the completion time of coflow C_k under DRF. We note that all the coflows will complete in the ascending order of their bottleneck demands, since DRF policy enforces an *equal* progress among coflows. We consider each time interval between two sequential coflows' completion, i.e., the time interval of $[F_{t-1}^{\mathrm{D}}, F_t^{\mathrm{D}}]$ and derive an *lower* bound of the coflow progress during this interval under DRF. Specifically, we have $P_t^* \leq e_{\max}/N_t$, where N_t denotes the largest count of coflows that share a common link within the coflow set of $\mathbb{C} = (C_t, \ldots, C_N)$. Based on this, for each coflow C_k, we derive an *lower* bound of its CCT under DRF, i.e., $F_k^{\mathrm{D}} \geq [\bar{d}_1 N_1 + N_2(\bar{d}_2 - \bar{d}_1) + \ldots N_k(\bar{d}_k - \bar{d}_{k-1})]/e_{\max}$.

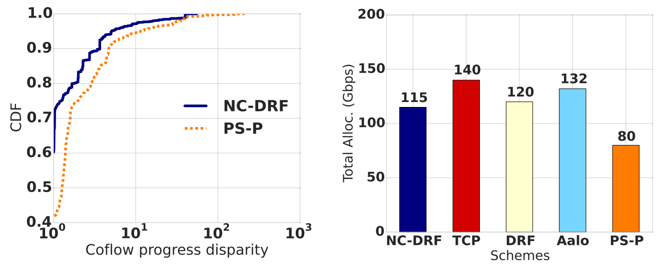
We further consider the CCTs under NC-DRF. We also note that, under our assumptions, the coflow completions under NC-DRF also follow the same order as that under DRF. Similarly, we consider the time interval of $[F_{t-1}, F_t]$ and derive a *upper* bound of the coflow progress under NC-DRF, i.e., $F_k \leq \bar{d}_1 N_1 + N_2(\bar{d}_2 - \bar{d}_1) + \ldots N_k(\bar{d}_k - \bar{d}_{k-1})$. We therefore obtain the CCT bound as shown in Theorem 1, i.e.,

$$\begin{aligned} F_k &\leq \bar{d}_1 N_1 + N_2(\bar{d}_2 - \bar{d}_1) + \ldots N_k(\bar{d}_k - \bar{d}_{k-1}) \\ &\leq e_{\max} F_k^{\mathrm{D}}. \end{aligned} \tag{7}$$

V. Evaluation

We evaluated NC-DRF through both trace-driven simulations and real-world implementations deployed with a 60-machine EC2 [16] cluster. We highlight following three points here:

- NC-DRF provides long-term fairness with guaranteed isolation between contending coflows. Specifically, with NC-DRF, coflows are only delayed by 68% on average as compared with the isolation-optimal DRF.
- NC-DRF dominates existing alternatives in long-term fairness, speeding the coflow completion. In particular, NC-DRF outperforms PS-P by 1.7× in terms of the average CCT.



(a) Coflow progress disparity at Scale (b) Average bandwidth allocation

Fig. 5: Characteristics of instantaneous fairness. (a) Distribution of coflow progress disparity, x-axis is in log scale. (b) Average utilization out of 300 Gbps availability.

- NC-DRF also dominates existing policies in terms of instantaneous fairness, providing smaller progress disparity across coflows. NC-DRF outperforms PS-P by 3.7× in terms of the maximum coflow progress disparity.

A. Trace-Driven Simulations

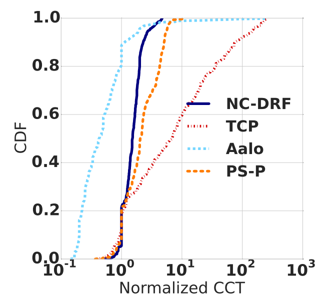
Workload. We use the production workload traces with 526 coflows in Coflow-Benchmark [17], which captures a realistic workload based on a one-hour Hive/MapReduce trace collected from a 3000-machine, 150-rack Facebook cluster. Particularly, the benchmark [17] reduces the original trace to *rack level* by combining all mappers (reducers) belonging to the same tenant in a rack into one rack-level mapper (reducer).

Setup. We abstract out the datacenter fabric as a 150×150 non-blocking switch, where an ingress (egress) port corresponds to a 1 Gbps uplink (downlink) of a rack. Therefore, the total bandwidth availability in the fabric is 300 Gbps. We compare NC-DRF against four scheduling schemes: per-flow fairness (TCP), FairCloud's PS-P policy [3] that seeks per-link fairness, DRF [1] and the performance-optimal scheduler Aalo [10]. We implemented these schemes along with NC-DRF on top of `CoflowSim` [31], the *de facto* simulator for coflow scheduling, comparing their instantaneous and long-term isolation of network service.

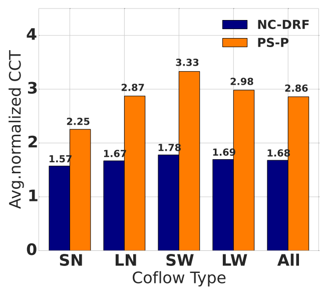
1) Instantaneous Isolation: We start by comparing the instantaneous isolation achieved by different policies. In particular, at each time instant, we calculate the progress disparity across all the coflows, as well as the entire bandwidth allocation in the fabric under each scheduler. Fig. 5 depicts the instantaneous performance comparisons in detail.

Coflow progress disparity. We have shown in Sec. IV that our proposed NC-DRF provides *long-term* isolation guarantees between contending coflows. A natural question is: How does NC-DRF perform when considering the *instantaneous coflow progress*? To answer this question, we measure the instantaneous isolation based on a metric called *coflow progress disparity*, which is defined, at each time instant, as the ratio between the maximum progress and the minimum progress across all the coflows, i.e.,

$$\text{Coflow progress disparity} = \frac{\max_k P_k}{\min_k P_k},$$



(a) Distribution of normalized CCT



(b) Average normalized CCT

Fig. 6: Characteristics of long-term isolation guarantee. (a) Distribution of normalized CCT. (b) Average normalized CCT in coflow bins.

TABLE I: Coflows binned by their lengths (**S**hort or **L**ong) and widths (**N**arrow or **W**ide) in the Coflow-Benchmark [17].

Bin	**SN**	**LN**	**SW**	**LW**
% of Coflows	60%	16%	12%	12%

where P_k represents the progress of coflow C_k given by Eq. (1). Intuitively, the smaller the coflow progress disparity, the better instantaneous isolation the policy can achieve. Particularly, the isolation-optimal DRF consistently keeps the coflow progress disparity equal to 1.

We measured the coflow progress disparity under NC-DRF and PS-P over time and depict the distribution in Fig. 5a. Here, we exclude TCP and Aalo from comparison due to their poor performance. We can observe that NC-DRF outperforms PS-P in terms of instantaneous fairness. Specifically, the coflow progress disparity under NC-DRF is less than 50 at 95% of time instants, whereas the corresponding value is more than 184 under PS-P.

Besides, the coflow progress disparity under both PS-P and Aalo shows a long-tail distribution. Specifically, the maximum coflow progress disparity under PS-P is more than 200, while it is less than 55 under NC-DRF. These results clearly indicate NC-DRF achieves better instantaneous isolation guarantee compared with PS-P, minimizing the progress differences between contending coflows. To summarize, NC-DRF outperforms PS-P by 3.7× in terms of the maximum coflow progress disparity.

Network Utilization. As we have illustrated in Sec. III, PS-P leads to unnecessary bandwidth wasting. We confirm this by measuring the total bandwidth utilization by all the coflows over time under different schemes. Fig. 5b compares the *average* bandwidth allocation out of 300 Gbps availability over time. First, we see that TCP is able to achieve the *highest* network utilization, as it performs scheduling at flow level and is not restricted by the coflow-level limitations. Second, PS-P performs the *worst*, resulting in the lowest average network utilization, even though it is *work conserving* [3]. We attribute this to its agnosticism to the coflow demand correlation, as the bandwidth in PS-P is assigned separately on each link, inevitably resulting in the *mismatching* of bandwidth allocation between each uplink (downlink) and its corresponding downlinks (uplinks). In contrast, NC-DRF is able to avoid this bandwidth wasting, providing average bandwidth utilization close to DRF policy.

TABLE II: Statistical summary of slowdown.

	TCP	**PS-P**	**NC-DRF**	**DRF**	**Aalo**
Min	1.00	1.00	**1.00**	1.00	1.00
Mean	117.94	9.47	**5.75**	3.36	5.40
95th	757	20.80	**11.14**	5.89	6.2416
Std.	246	6.75	**3.64**	1.52	57.67

2) Long-Term Isolation: We next evaluate the *long-term* performance of different schedulers in achieving isolation guarantees. In particular, we replayed 526 coflows in the trace [17] and used two metrics throughout our evaluation: normalized CCT and shuffle slowdown.

- *Normalized CCT* is defined, for each coflow, as the CCT under the compared scheduler normalized by that under DRF:
$$\text{Normalized CCT} = \frac{\text{Compared CCT}}{\text{CCT under DRF}}.$$
Intuitively, the smaller the normalized CCT, the faster coflows will complete under the compared scheduler, and hence the better long-term isolation guarantee it achieves.
- *Shuffle slowdown* is defined, for each coflow, as the CCT under the compared scheduler normalized by the minimum CCT, i.e., its bottleneck's completion time when running alone in the fabric:
$$\text{Slowdown} = \frac{\text{Compared CCT}}{\text{Minimum CCT}}.$$

Additionally, to better understand the performance impact on different coflows, we categorize coflows into four bins based on their shuffle types. Specifically, we say a coflow is *small* (*long*) if its largest flow is less (greater) than 5 MB, and *narrow* (*wide*) if it consists of *less* (more) than 50 flows [2], [10], [22]. Table I summarizes the distribution of binned coflows.

Normalized CCT. Fig. 6a depicts the distribution of normalized CCT under different schemes. We can see that TCP performs worst, arbitrarily delaying the coflow completion. We also observe that the performance-optimal Aalo can dramatically speed the coflow completion, however, provides no isolation guarantee, resulting in normalized CCT more than 100. More importantly, NC-DRF outperforms PS-P in that it leads to shorter normalized CCT.

To better illustrate the difference between NC-DRF and PS-P, we compare the average normalized CCT between these two schemes in four coflow bins as shown in Table I and plot the results in Fig. 6b. We see that NC-DRF consistently outperforms PS-P, with smaller normalized CCT in all the bins. Moreover, NC-DRF outperforms PS-P by 1.7× in terms of the average normalized CCT. It is worth noticing that with NC-DRF, coflows are only delayed by 68% on average as compared with the isolation-optimal DRF—a strong evidence that NC-DRF is able to achieve long-term isolation guarantee.

Shuffle slowdown. Table II gives the minimum, average, and the 95th percentile slowdown, as well as the standard deviation measured under different schedulers. We can see

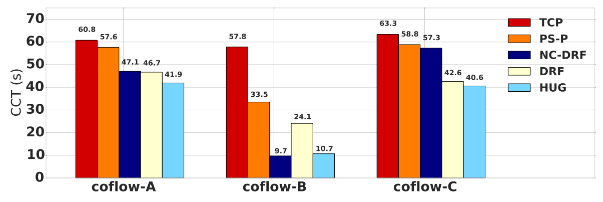


Fig. 7: CCT of three coflows in a 60-machine cluster.

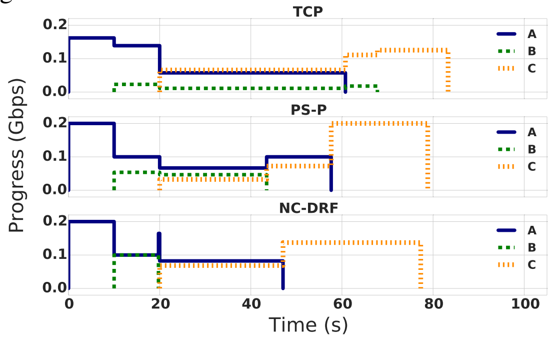


Fig. 8: Coflow progress measured over time.

that NC-DRF consistently outperforms PS-P under all the four metrics. To summarize, in terms of the shuffle slowdown, NC-DRF outperforms PS-P by $1.65\times$ on average, and by $1.87\times$ at the 95th percentile.

B. EC2 Deployment

Implementation. We have prototyped NC-DRF in `Python` with a master-slave architecture. In particular, the `master` identifies coflows and makes scheduling decisions following the logic of Algorithm 1; a `slave` runs in a cluster machine as a daemon program and enforces the specified flow transmission rate using Linux's `tc` and `htb qdisc` tools to shape the flow rates. Specifically, upon arrival, a coflow registers to the `master` and indicates the amount of data in each flow through a public NC-DRF API. The `master`, after receiving this registration, runs Algorithm 1 and provides a new allocation for active coflows and notifies the flow transmission rates to the corresponding `slaves` for local enforcement. Each `slave` also updates its status as heartbeat messages with the `master` periodically, which allows the `master` to quickly respond to coflow.

Cluster deployment. We performed experiments in a 60-node Amazon EC [16] cluster. For each node, we used a `t2.large` EC2 instance with 2 cores and 8 GB RAM. For simplicity, we configured 200 Mbps as the bandwidth capacity to each uplink/downlink.

Micro-benchmark. To micro-benchmark the behavior of NC-DRF in a more controlled manner, we ran three coflows, each having endpoints on all 60 machines with different communication patterns. Table III summarizes the configurations of three coflows. Particularly, for coflow-A, we evenly divide its endpoints into 10 groups. Each group performs an all-to-all shuffle, performing 6×6 communication. In total, coflow-A consists of 360 flows. Coflow-B has 60 flows following a *pairwise one-to-one* communication pattern between machine i and machine $i + 30$, where $1 \leq i \leq 30$. Coflow-C also has 60 flows following the same communication pattern, where machine j communicates with machine $j + 15$, for all $1 \leq j \leq 15$ and $30 \leq j \leq 45$. In total, we have 480 flows in three coflows, each randomly configured its transferred data size between 30 MB and 100 MB. In this setting, coflow-A represents a large job, whereas the other two coflows are considered small. Coflow-A, B, and C arrive at 0 s, 10 s, and 20 s, respectively.

TABLE III: Summary of coflow information in testbed.

	Commun. Pattern	# of Flows	Arrival Time
coflow-A	all-to-all	360	0 s
coflow-B	pairwise one-to-one	60	10 s
coflow-C	pairwise one-to-one	60	20 s

*1) **Coflow Completion Time:*** Fig. 7 illustrates the CCTs of three coflows under three schedulers. We can observe that NC-DRF consistently outperforms the alternative TCP and PS-P, resulting in shorter CCTs of all the three coflows. Even compared with the isolation-optimal HUG and DRF which assume complete coflow knowledge a priori, NC-DRF still provides shorter CCT for coflow-B.

*2) **Coflow Progress:*** In Fig. 8, we depict the progress of each coflow under TCP, PS-P and NC-DRF over time. We see that NC-DRF provides *nearly equal* progress between coflow-A and coflow-B during 10-20 second, as well as coflow-A and coflow-C during 20-47 second. In other words, NC-DRF can not only achieve the long-term isolation by reducing the coflow completion time, but also strive for the instantaneous equal progress across coflows. Notably, in our testbed experiment, we do not enforce the equal size of flows, we instead randomly choose the flow size. On the other hand, we can clearly observe that neither TCP nor PS-P can enforce such equal progress between coflows. We attribute this advantage to NC-DRF's awareness of the coflow demand correlation, with which the scheduler can expect long-term isolation guarantee.

VI. Conclusion

In this paper, we have studied non-clairvoyant coflow scheduling to isolating their completions, which has insofar received little attention. We for the first time have clarified the definition of isolation guarantee for non-clairvoyant coflow scheduling. We have also proposed a new coflow scheduler, called Non-Clairvoyant DRF (NC-DRF), to provide isolation guarantees between contending coflows, without prior knowledge of coflow size. NC-DRF outperforms the alternatives by being aware of the coflow-level communication patterns. We have shown that under some practical assumptions, with NC-DRF, the CCT of each coflow is bounded by a small constant factor of that under DRF. Both trace-driven simulations and EC2 deployments have confirmed that NC-DRF outperforms existing alternatives and achieves long-term isolation guarantee.

We recognize that NC-DRF does not ensure the optimal long-term isolation guarantee, which we leave for future work. Specifically, theoretical investigations should be performed in the future, such as how to minimize the constant ratio bound of the achieved long-term isolation compared with fair schedulers.

References

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

[2] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "HUG: Multi-resource fairness for correlated and elastic demands," in *USENIX NSDI*, 2016.

[3] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *ACM SIGCOMM*, 2012.

[4] M. Chowdhury, "Coflow: A networking abstraction for distributed data-parallel applications," Ph.D. dissertation, University of California, Berkeley, 2015.

[5] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," in *ACM SIGCOMM*, 2012.

[6] A. Shieh, S. Kandula, A. G. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *USENIX NSDI*, 2011.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.

[8] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *ACM CoNEXT*, 2010.

[9] W. Wang and A.-L. Jin, "Friends or foes: Revisiting strategy-proofness in cloud network sharing," in *IEEE ICNP*, 2016.

[10] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *ACM SIGCOMM*, 2015.

[11] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: Practical network performance isolation at the edge," in *USENIX NSDI*, 2013.

[12] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward automatically identifying and scheduling coflows in the dark," in *ACM SIGCOMM*, 2016.

[13] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online." in *USENIX NSDI*, 2010.

[14] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *ACM SIGMOD*, 2014, pp. 147–156.

[15] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *USENIX OSDI*, 2004.

[16] "Amazon ec2," http://aws.amazon.com/ec2.

[17] M. Chowdhury, "Coflow-Benchmark," https://goo.gl/szsBQE.

[18] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *ACM SIGCOMM*, 2009.

[19] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM*, 2009.

[20] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM*, 2014.

[21] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *ACM SIGCOMM*, 2015.

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

[23] "Apache tez," http://tez.apache.org.

[24] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica, "Pacman: Coordinated memory caching for parallel jobs," in *USENIX NSDI*, 2012.

[25] M. Zaharia, M. Chowdhury, T. Das, A. Dave, 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 *USENIX NSDI*, 2012.

[26] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *EuroSys*, 2007.

[27] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *ACM SIGCOMM*, 2011.

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

[29] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[30] "Fair coflow scheduling without prior knowledge," https://goo.gl/HRmFKC, Tech. Rep., 2017.

[31] "Coflowsim," https://github.com/coflow/coflowsim.