Efficient Coflow Transmission for Distributed Stream Processing

Wenxin Li*, Xu Yuan†, Wenyu Qu‡§, Heng Qi¶, Xiaobo Zhou‡§, Sheng Chen‡§, Renhai Xu‡§
*Hong Kong University of Science and Technology, Hong Kong
†University of Louisiana at Lafayette, USA
‡College of Intelligence and Computing, Tianjin University, China
§Tianjin Key Laboratory of Advanced Networking (TANK)
¶Dalian University of Technology, China

***Abstract*—Distributed streaming applications require the underlying network flows to transmit packets continuously to keep their output results fresh. These results will become stale if no updates come, and their staleness is determined by the slowest flow. At this point, *coflows* can be semantically comprised. Hence, efficient coflow transmission is critical for streaming applications. However, prior coflow-based solutions have significant limitations. They use a one-shot performance metric—CCT (coflow completion time), which cannot continuously reflect the staleness of the output results for a streaming application.**

To this end, we propose a new performance metric—*coflow age* (CA), for coflows generated by distributed streaming applications. The CA tracks the *longest time-since-last-service* among all flows in a coflow. In such a context, we consider a data center network with multiple coflows that continuously transmit packets between their source-destination pairs and address the problem of minimizing the average long-term CA while simultaneously satisfying the throughput constraints from the coflows. To solve this problem efficiently, we design a randomized algorithm and a drift-plus-age algorithm, and show that they can make the average long-term CA to achieve nearly two times and arbitrarily close to the optimal value, respectively. Through extensive simulations, we further demonstrate that both of the proposed algorithms can significantly reduce the CA of coflows, without violating the throughput requirement of any coflow, when compared to the state-of-the-art solution.

I. Introduction

An emerging trend in big data processing is to extract meaningful results from continuous data streams (e.g., sensor readings and online social media) with distributed computation running on a large data center [1–3]. Because the extracted results are usually used by real-time business-critical decision algorithms, their value will decrease as time goes by. On the other hand, distributed stream computation will generate a group of flows to transmit packets (e.g., key-value records [4]) among different machines across the data center network. Each packet carries the most up-to-date information. From the perspective of streaming applications, they wish to *receive packets from every flow continuously*, so as to output fresh results to the real-time decision algorithms.

The *coflow* has recently become an elegant model to specifically abstract a semantical-related collection of flows for a broad spectrum of applications [5]. Different applications impose different semantics on their flows. For example, in a batch processing application, only completing all flows in a coflow is meaningful. In contrast to batch processing, flows generated by streaming applications have no strict concept of completion, and they must transmit packets continuously to prevent them from being straggling behind. In other words, the results extracted by a streaming application at a certain time can be considered fresh only if all its flows can transmit a new packet. Therefore, to optimize the performance of a streaming application, we need to schedule packet transmissions at the level of coflow rather than individual flows.

However, prior solutions (e.g., [6–12]) only focus on coflows generated by batch processing applications. Moreover, they cannot be adopted in streaming applications due to the one-shot performance metric they used, namely CCT (coflow completion time). The CCT is defined for a static set of packets and can only be figured out after all packet transmissions finish. Apparently, CCT is unable to continuously reflect the staleness of a streaming application's output results.

Motivated by this situation, we present a new performance metric, called as *coflow age* (CA), for coflows generated by distributed streaming applications. The CA of a coflow is defined as the maximum age of all its flows, where the age of an individual flow represents the time elapsed since the most recent packet is received by the destination. More specifically, if a packet is successfully delivered to its destination, the age of the corresponding flow drops to zero immediately and grows linearly otherwise. The metric CA[1] essentially captures the slowest flow in a coflow each time, i.e., the flow whose destination did not receive a new packet for the longest time. It is a constantly evolving measurement and is aligned with the needs of streaming applications.

For a better intuition of the CA, we use an example in Fig. 1, where there is a `WordCount` streaming application that calculates the count of each distinct word in the input stream over time. This application has one `split` task (on server `S1`) and two `count` tasks (on servers `S2` and `S3`

[1]It should be noted that the concept of CA is borrowed from the age-of-information (AoI) which is an important performance metric in wireless networks [13, 14]. The AoI measures the time elapsed since the newest data was generated at the source rather than since the most recent data was received by the destination. Even though both AoI and CA are age-based metrics, CA is more relevant. The reason is that the interest of a streaming application is in how long it did not obtain new output results, rather than in continuously monitoring the freshness of the input data.

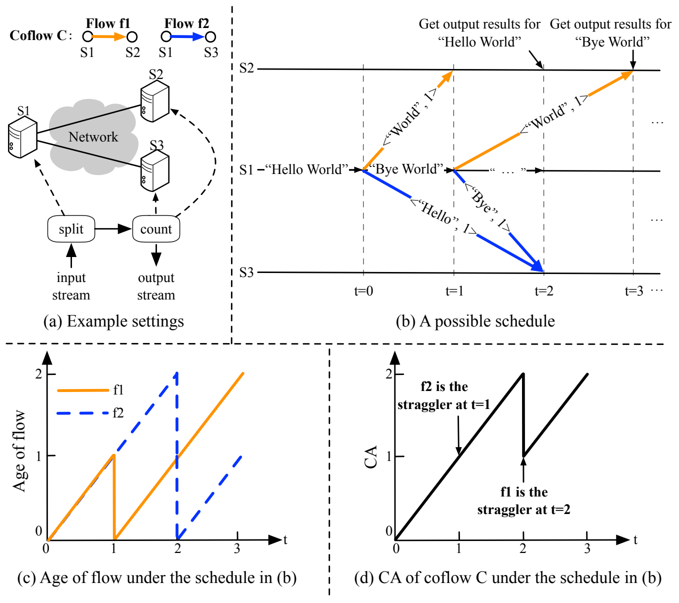

Coflow C:
Flow f1
Flow f2
S1
S2
S3
Network
split
count
input stream
output stream
(a) Example settings
Get output results for "Hello World"
Get output results for "Bye World"
S1 –"Hello World"
"Bye World"
<"World", 1>
<"Hello", 1>
<"Bye", 1>
t=0
t=1
t=2
t=3
(b) A possible schedule
Age of flow
f1
f2
(c) Age of flow under the schedule in (b)
CA
f2 is the straggler at t=1
f1 is the straggler at t=2
(d) CA of coflow C under the schedule in (b)

Fig. 1. An example for illustrating that the metric CA can reflect the staleness of the output results of a streaming application (i.e., `WordCount`).

respectively). It generates a coflow `C` when processing the unbounded input stream, which has two flows `f1` (`S1`→`S2`) and `f2` (`S1`→`S3`). Suppose `f1` and `f2` transmit packets like Fig. 1(b). Then, at $t = 1$, the CA of coflow `C` is 1 and the complete output results for "Hello World" cannot be obtained, because `f2`'s destination `S3` has not received a new packet since $t = 0$. The CA of `C` will keep increasing until $t = 2$, because at $t = 2$, `S3` receives the required packet for counting words in "Hello World". Nevertheless, the CA of `C` only decreases to 1 at $t = 2$. The reason is that `f1` becomes straggling behind at this time with respect to the output for "Bye World". As time goes by, the CA of `C` will constantly evolve. Needless to say, persistently pushing the CA of `C` to a low value can improve the stream processing efficiency for the `WordCount` application.

To optimize CA, the scheduler must decide *which* coflows to transmit packets each time. Given that there are usually more than one coflows sharing the same data center network, some coflows may transmit packets repeatedly while others less often. In fact, the frequency at which the output results are produced is also important for streaming applications [15], implying that the coflows may have throughput requirements. So, we ponder a fundamental question: *can we design coflow scheduling algorithms that minimize the CA while simultaneously satisfying throughput requirements of coflows?*

In this paper, we provide a cautiously optimistic answer by developing two efficient online algorithms. To be particular, we first develop a *stationary randomized* algorithm, which selects each flow for transmission purely according to a pre-computed scheduling probability. This algorithm is practical yet straightforward, and it can achieve a solution that nearly equals to two times the optimal one. By taking advantage of additional information (e.g., current age of flows) when selecting flows for transmission, we further develop a *drift-plus-age* algorithm. This algorithm is designed to reduce the average CA while controlling the growth of a well defined Lyapunov function to satisfy the throughput requirements of coflows. We have proved that the drift-plus-age algorithm can arbitrarily approach the optimal solution. We have conducted extensive trace-driven simulations to demonstrate the effectiveness of our proposed algorithms in optimizing the CA. Compared with the state-of-the-art solution [7] which is age-agnostic, the stationary randomized and the drift-plus-age algorithms can reduce the average CA by 31.8% and 47.5% on average of 100 tries respectively while satisfying the throughput requirements of all coflows.

In summary, the main contributions of this paper include:

- We present a new performance metric CA to quantify the transmissions of coflows generated by distributed streaming applications.
- We study and formulate the problem of minimizing the average long-term CA while guaranteeing the throughput requirements of coflows. Moreover, we present two online algorithms (i.e., a stationary randomized algorithm and a drift-plus-age algorithm) to solve the problem.
- We prove that the proposed two algorithms can achieve nearly two times and arbitrarily close to the optimum respectively, by conducting rigorous theoretical analysis.
- We conduct comprehensive trace-driven simulations to evaluate the performance of our proposed algorithms, in terms of reducing the average CA and guaranteeing throughput requirements of coflows.

The rest of this paper is organized as follows. In Section II, we develop the mathematical model and present our problem formulation. We show the design details of the proposed two algorithms in Section III. Extensive simulations are presented in Section IV. Section V discusses the related work and Section VI concludes this paper.

II. Modeling and Problem Formulation

We consider the data center network as a non-blocking switch where the bottlenecks only take place at the ingress and egress ports of the switch, corresponding to the incoming and outgoing links at each server. Such a network is reasonable yet practical and has been widely adopted in coflow studies [6, 7, 11]. The network is shared by a set of coflows that are submitted by multiple streaming applications. Each coflow contains a set of flows that continuously transmit packets between two fixed groups of servers. As streaming applications are typically long-running, the source and destination nodes of each flow can be known a prior [5], while the packet arrivals are uncertain.

Concerning the above scenario, our goal is to find the optimal scheduling strategy we can apply to the packets of coflows to minimize the CA while satisfying throughput requirements of coflows.

A. Notation

We first introduce the important notations used throughout this paper. We denote $\mathcal{M} = \{1, 2, \ldots, M\}$ as the set of servers in the data center network and consider a discrete time mode, where the time is divided into T time slots. We denote $\mathcal{T} = \{0, 1, \ldots, T-1\}$ as the set of time slots. The granularity of each time slot is typically the time taken to transmit a single MTU[2]-sized packet over the fastest link in the network [16]. For simplicity, in each time slot $t \in \mathcal{T}$, we assume each server $i \in \mathcal{M}$ can transmit or receive at most one packet through its outgoing or incoming link. We denote $\mathcal{K} = \{1, 2, \ldots, K\}$ as the set of coflows. Each coflow $k \in \mathcal{K}$ consists of N_k flows, among which the n-th flow is denoted by $f_{k,n}$. The source and destination nodes of flow $f_{k,n}$ are denoted as $s_{k,n}$ and $d_{k,n}$ ($s_{k,n}, d_{k,n} \in \mathcal{M}$), respectively.

To indicate the scheduling strategy, we denote $x_{k,n}(t)$ as if a scheduler selects $f_{k,n}$ for transmission at time t. Note that there may be no packet get ready for transmission at time t even when flow $f_{k,n}$ is selected for transmission (i.e., $x_{k,n}(t) = 1$). So, we define $p_{k,n}$ as the probability that a packet of flow $f_{k,n}$ arrives at its source $s_{k,n}$. We assume that this probability $p_{k,n}$ does not change with time, but may differ for different flows. It is worth noting that such probability-based modeling method may be simple, but it suffices to capture the uncertain property in packet arrivals and has also been widely adopted in existing literature [16, 17].

Let $y_{k,n}(t)$ be a random variable representing whether a packet is delivered from $s_{k,n}$ to $d_{k,n}$ at time slot t. Specifically, if flow $f_{k,n}$ is selected for transmission at time t, i.e., $x_{k,n}(t) = 1$, then $y_{k,n}(t) = 1$ with probability $p_{k,n}$ and $y_{k,n}(t) = 0$ with probability $1 - p_{k,n}$. On the other hand, if $x_{k,n}(t) = 0$, then $y_{k,n}(t) = 0$ with probability one. By applying the law of iterated expectations, we have

$$\mathbb{E}\{y_{k,n}(t)\} = p_{k,n}\mathbb{E}\{x_{k,n}(t)\}. \tag{1}$$

B. Mathematical model

We now present the mathematical model for our problem.

Coflow age (CA): As mentioned in the above section, CA is a constantly evolving measurement. As a performance metric, CA is more aligned with the essential needs of streaming applications to quantify how long their output results have not been updated. Before presenting CA, we first introduce the age of an individual flow, which has a tractable form of evolution and can track the time since the last packet is served. To be particular, let $a_{k,n}(t)$ denote the instantaneous age of flow $f_{k,n}$ at the end of time slot t. The evolution of $a_{k,n}(t)$ is shown as follows

$$a_{k,n}(t) = \begin{cases} 0, & \text{if } y_{k,n}(t) = 1, \\ a_{k,n}(t-1) + 1, & \text{otherwise,} \end{cases} \tag{2}$$

which means that if a packet of $f_{k,n}$ is successfully delivered to $d_{k,n}$ at the end of time slot t, then the age of this flow will drop to 0; otherwise, $a_{k,n}(t) = a_{k,n}(t-1) + 1$, as the last received packet is one time slot older.

We now can define the instantaneous CA of coflow k at the end of time slot t as the maximum age of all its flows.

$$A_k(t) = \max_{n=1,\ldots,N_k} a_{k,n}(t). \tag{3}$$

[2]MTU (Maximum Transmission Unit) is the largest packet size that can be sent over a network connection. For instance, the MTU of an Ethernet connection is typically 1500 bytes.

However, from a realistic world's view, the age of a flow cannot be observed at every time instant, as the length of each time slot cannot be infinitely small. This motivates us to turn to *long-term* CA as follows

$$\bar{A}_k = \max_{n=1,\ldots,N_k} \left\{ \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{a_{k,n}(t)\} \right\}. \tag{4}$$

The term $\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{a_{k,n}(t)\}$ captures the long-term age of flow $f_{k,n}$. Thus, Eq. (4) measures the maximum long-term age of the flows in coflow k. The expectation is with respect to the randomness in packet arrivals and the scheduling policy. It should be noted that the long-term CA is not necessarily in conflict with the instantaneous CA, and a lower long-term CA can lead to a lower instantaneous CA on average across all time slots.

Throughput constraints: In real-world scenarios, streaming applications can impose constraints on the frequencies at which their output results are produced [15, 18]. For example, a streaming application may monitor the user logs continuously to obtain the number of clicks on a recommended URL once every few seconds. To indicate such frequency requirements, we consider that the flows in the same coflow have identical throughput requirements. The rationale here is that the CA of a coflow can be minimally influenced when considering identical throughput requirements for its flows. Specifically, we denote q_k as the throughput requirement for each of the flows in coflow k. Then, we have

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{y_{k,n}(t)\} \geq q_k, \forall k \in \mathcal{K}, \forall n \in \{1, \ldots, N_k\}. \tag{5}$$

The L.H.S. of the above inequality is the *long-term throughput* of flow $f_{k,n}$. Hence, Eq. (5) essentially means that the long-term throughput of flow $f_{k,n}$ should be at least q_k.

Scheduling constraints: When scheduling the packets of different coflows, the constraints associated with the incoming and outgoing links on each server impose that

$$\sum_{k=1}^{K} \sum_{n=1}^{N_k} x_{k,n}(t) e(s_{k,n} = i) \leq 1, \forall t \in \mathcal{T}, \forall i \in \mathcal{M}, \tag{6}$$

$$\sum_{k=1}^{K} \sum_{n=1}^{N_k} x_{k,n}(t) e(d_{k,n} = i) \leq 1, \forall t \in \mathcal{T}, \forall i \in \mathcal{M}. \tag{7}$$

Note here $e(\chi) = 1$ if event χ is true and $e(\chi) = 0$ otherwise. Eq. (6) means that in any given time slot t, the scheduler can select at most one flow for transmission through the outgoing link of server i. Similarly, Eq. (7) ensures that at most one flow can be selected for transmission through the incoming link of server i in each time slot t.

Decision variable: For each flow $f_{k,n}$ at each time slot t, the decision variable $x_{k,n}(t)$ can only take 0 or 1,

$$x_{k,n}(t) \in \{0, 1\}, \forall k \in \mathcal{K}, \forall n \in \{1, \ldots, N_k\}, \forall t \in \mathcal{T}. \tag{8}$$

C. Problem formulation

With the above mathematical model, we study the problem of minimizing the average CA while satisfying the throughput constraints of coflows. We denote this problem as **P1** which is formulated as follows:

$$\min_{\{x_{k,n}(t)\}} \frac{1}{K}\sum_{k=1}^{K}\bar{A}_k \tag{9}$$
$$\text{s.t. Eqs. } (1), (5), (6), (7), (8).$$

Clearly, the objective in Eq. (9) is to minimize the average long-term CA across all coflows. It should be noted that it is inherently a hard task to solve the problem **P1**, due to the following challenges. First, as **P1** is a long-term stochastic optimization, the control decisions (i.e., $x_{k,n}(t)$) at current time slot will impact that at future time slots. Second, minimizing the average age in stochastic scheduling problems with more than one flow is NP-hard in general [14], while **P1** involves multiple coflows and each coflow contains a set of flows. Third, the packet arrivals are highly dynamic and uncertain with much randomness. Hence, efficient online scheduling algorithms are highly desired here to solve the problem **P1**.

III. Scheduling Algorithms

In response to the challenges of solving the problem **P1**, we present two online algorithms. The first algorithm solves the problem **P1** over the class of stationary randomized policies and returns a solution that approximately equals to two times the optimal solution. In contrast, the second algorithm is derived using Lyapunov optimization techniques and can arbitrarily approach the optimal solution.

A. Stationary randomized algorithm

The key idea of this algorithm is to find the best scheduling policy among the class of stationary randomized policies when solving the problem **P1**. More specifically, in the algorithm design, we first transform the original problem **P1** into a convex optimization where the decision variables are a set of scheduling probabilities. Each possible set of scheduling probabilities can characterize a stationary randomized policy. However, even we have the solution for this convex optimization, it does not give a scheduling strategy for **P1** directly. Therefore, we need another technique, which is usually called sampling, to get a feasible solution.

To ease the presentation, let Π denote the class of stationary randomized policies, where each $R \in \Pi$ is a scheduling policy that selects flow $f_{k,n}$ for transmission with probability $\theta_{k,n} \in (0, 1]$. Therefore, each policy R can actually be characterized by the set of scheduling probabilities $\{\theta_{k,n}, \forall k, \forall n\}$, where $\theta_{k,n} = \mathbb{E}\{x_{k,n}(t)\}, \forall k, \forall n, \forall t$. With the scheduling probabilities, we have the following theorem.

Theorem 1: *Under a $R \in \Pi$ with scheduling probabilities $\{\theta_{k,n}, \forall k, \forall n\}$, the long-term throughput and the expected time-averaged age of each flow can be reformulated as*

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{y_{k,n}(t)\} = p_{k,n}\theta_{k,n}, \tag{10}$$

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{a_{k,n}(t)\} = \frac{1}{p_{k,n}\theta_{k,n}}. \tag{11}$$

Proof: Let $I_{k,n}[l]$ be a random variable representing the number of time slots between the $(l-1)$-th and l-th packets delivered for flow $f_{k,n}$. Under the policy R, each packet of $f_{k,n}$ can successfully be delivered with probability $p_{k,n}\theta_{k,n}$. In other words, for each $f_{k,n}$ at any t, its destination $d_{k,n}$ receives a packet from the source $s_{k,n}$ only if $f_{k,n}$ is scheduled and there exists a packet at $s_{k,n}$. The probability when $I_{k,n}[l]$ equals to η $(= 1, 2, \dots)$ is $\mathbb{P}\{I_{k,n}[l] = \eta\} = p_{k,n}\theta_{k,n}(1 - p_{k,n}\theta_{k,n})^{\eta-1}$. It should be noted that under the policy R, the sequence of packet delivers is a renewal process. Thus, we can use renewal theory to derive Eq. (10) and Eq. (11). In particular, we have the following equalities

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{y_{k,n}(t)\} = \frac{1}{\mathbb{E}\{I_{k,n}[l]\}} = p_{k,n}\theta_{k,n}, \tag{12}$$

$$\begin{aligned}\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{a_{k,n}(t)\} &= \frac{\mathbb{E}\{I_{k,n}^2[l]\}}{2\mathbb{E}\{I_{k,n}[l]\}} + \frac{1}{2} \\ &= \frac{1}{p_{k,n}\theta_{k,n}}.\end{aligned} \tag{13}$$

where Eq. (12) and Eq. (13) follow from the elementary renewal theorem and the generalization for renewal-reward processes [19], respectively. ■

Given the above theorem, we now can formulate the following optimization problem, denoted as **P2**:

$$\min_{\{\theta_{k,n}\}} \frac{1}{K}\sum_{k=1}^{K}\max_{n=1,\dots,N_k}\frac{1}{p_{k,n}\theta_{k,n}} \tag{14}$$

$$\text{s.t. } p_{k,n}\theta_{k,n} \geq q_k, \forall k \in \mathcal{K}, \forall n \in \{1,\dots,N_k\}, \tag{15}$$

$$\sum_{k=1}^{K}\sum_{n=1}^{N_k}\theta_{k,n}e(s_{k,n}{=}i) \leq 1, \forall i \in \mathcal{M}, \tag{16}$$

$$\sum_{k=1}^{K}\sum_{n=1}^{N_k}\theta_{k,n}e(d_{k,n}{=}i) \leq 1, \forall i \in \mathcal{M}, \tag{17}$$

$$0 < \theta_{k,n} \leq 1, \forall k \in \mathcal{K}, \forall n \in \{1,\dots,N_k\}, \tag{18}$$

where Eq. (14) is derived by substituting Eq. (11) into Eq. (9). Substituting Eq. (10) into Eq. (5) results in Eq. (15). Eqs. (16) and (17) are obtained by substituting $\theta_{k,n} = \mathbb{E}\{x_{k,n}(t)\}$ into Eqs. (6) and (7), respectively. Finally, Eq. (18) depicts the value range for the decision variable $\theta_{k,n}$.

Problem **P2** is a convex optimization because its objective function and constraint functions are all convex with respect to $\theta_{k,n}$. It can be solved with standard convex optimization tools [20] or by using Lagrange multipliers and KKT conditions [21]. Since $\theta_{k,n}$ only indicates scheduling probability for a flow $f_{k,n}$, the solution of **P2** may not give a feasible scheduling strategy for the original problem **P1**. Hence, we propose to use another technique—*sampling*.

The whole procedure of the stationary randomized algorithm is summarized in Algorithm 1. At the beginning of each time slot t, Algorithm 1 starts by initializing a variable U_i which is used to indicate whether the incoming link of server $i \in \mathcal{M}$ has been occupied (Step 1). Then, in the *for* loop (Step 2-5), it selects a flow for each outgoing link independently. Specifically, it samples a flow $f_{k,n}$ for the outgoing link

Algorithm 1 Stationary Randomized Algorithm

```
Input: Coflow information: $p_{k,n}, s_{k,n}, d_{k,n}, q_k, \forall k, \forall n$
Output: Scheduling strategy: $x_{k,n}(t), \forall k, \forall n, \forall t$
1: At the beginning of each time slot $t$, initialize $U_i = 1, \forall i$.
2: for $i = 1, \ldots, M$ do
3:     Sample a flow $f_{k,n}$ with probability $\theta_{k,n}$ from $\{f_{k,n} : e(s_{k,n} = i), \forall k, \forall n\}$. $\{\theta_{k,n}\}$ is the optimal solution of problem P2, which can be calculated off-line.
4:     If $U_{d_{k,n}} = 0$, repeat step 3. Otherwise, set $x_{k,n}(t) = 1$ and update $U_{d_{k,n}} = 0$.
5: end for
6: return $\{x_{k,n}(t)\}$
```

of each server i with probability $\theta_{k,n}$, and the sample set is the set of flows that share a same source node i, i.e., $\{f_{k,n} : e(s_{k,n} = i), \forall k, \forall n\}$. Note that if the sampled flow cannot be accommodated by the incoming link of its relevant destination node, Algorithm 1 will repeat the sampling process until it finds a flow that can be accommodated. To analyze the performance of Algorithm 1, we first give *a lower bound* of the problem **P1** through Theorem 2. Then, we use Theorem 3 to prove that our Algorithm 1 can approximately achieve two times the optimal solution to the problem **P1**.

***Theorem** 2:* Let the optimal objective value of the original problem **P1** be denoted by O^*_{P1}. The lower bound of O^*_{P1} can then be characterized by the optimal objective value O^*_{P3} of the following problem **P3**:

$$\min_{\{x_{k,n}(t)\}} \frac{1}{2K}\sum_{k=1}^{K} \max_{n=1,\ldots,N_k}\left\{\frac{1}{\hat{q}_{k,n}}-1\right\}, \tag{19}$$

$$\text{s.t. } \hat{q}_{k,n} \geq q_k, \forall k, \forall n \in \{1,\ldots,N_k\}, \tag{20}$$

$$\text{Eqs. } (1),(6),(7),(8).$$

where $\hat{q}_{k,n} = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{y_{k,n}(t)\}$. This means that for any setting $(M, K, N_k, s_{k,n}, d_{k,n}, p_{k,n}, q_k)$, $O^*_{P1} \geq O^*_{P3}$.

Proof: For any feasible solution of problem **P1**, define $Y_{k,n}(T) = \sum_{t=0}^{T-1} y_{k,n}(t)$ as the total number of delivered packets for flow $f_{k,n}$ during the T time slots. Recall that $I_{k,n}[l]$ is defined as a random variable representing the number of time slots between the $(l-1)$-th and l-th delivered packets of flow $f_{k,n}$. Hence, for each $f_{k,n}$, we have

$$T = \sum_{l=1}^{Y_{k,n}(T)} I_{k,n}[l] + \gamma_{k,n}, \tag{21}$$

where $\gamma_{k,n}$ is the number of remaining time slots after the last delivered packet of $f_{k,n}$.

Considering the evolution behavior of $a_{k,n}(t)$, we can infer that $a_{k,n}(t)$ evolves as $\{0, 1, \ldots, I_{k,n}[l]-1\}$ and it will repeat this pattern throughout the entire time-horizon including the last $\gamma_{k,n}$ time slots. Hence, we can express the time-averaged age of flow $f_{k,n}$ as follows

$$\frac{1}{T}\sum_{t=0}^{T-1} a_{k,n}(t) = \frac{1}{T}\left[\sum_{l=1}^{Y_{k,n}(T)}\frac{(I_{k,n}[l]-1)I_{k,n}[l]}{2}+\frac{(\gamma_{k,n}-1)\gamma_{k,n}}{2}\right]$$
$$= \frac{1}{2}\left[\frac{Y_{k,n}(T)}{T}\frac{1}{Y_{k,n}(T)}\sum_{l=1}^{Y_{k,n}(T)} I^2_{k,n}[l] + \frac{\gamma^2_{k,n}}{T} - 1\right], \tag{22}$$

where the last equality is obtained by using Eq. (21). Let the sample mean of $I_{k,n}$ and $I^2_{k,n}$ be defined by the following two equalities, respectively.

$$\bar{W}[I_{k,n}] = \frac{1}{Y_{k,n}(T)}\sum_{l=1}^{Y_{k,n}(T)} I_{k,n}[l], \tag{23}$$

$$\bar{W}[I^2_{k,n}] = \frac{1}{Y_{k,n}(T)}\sum_{l=1}^{Y_{k,n}(T)} I^2_{k,n}[l]. \tag{24}$$

Then, substituting Eq. (24) into Eq. (22) and applying Jensen's inequality, we yield

$$\frac{1}{T}\sum_{t=0}^{T-1} a_{k,n}(t) \geq \frac{1}{2}\left[\frac{Y_{k,n}(T)}{T}(\bar{W}[I_{k,n}])^2 + \frac{\gamma^2_{k,n}}{T} - 1\right]$$
$$= \frac{1}{2}\left[\frac{1}{T}\frac{(T-\gamma_{k,n})^2}{Y_{k,n}(T)} + \frac{\gamma^2_{k,n}}{T} - 1\right], \tag{25}$$

By minimizing $\frac{1}{2}\left[\frac{1}{T}\frac{(T-\gamma_{k,n})^2}{Y_{k,n}(T)} + \frac{\gamma^2_{k,n}}{T} - 1\right]$ with respect to the variable $\gamma_{k,n}$, we have

$$\frac{1}{T}\sum_{t=0}^{T-1} a_{k,n}(t) \geq \frac{1}{2}\left(\frac{T}{Y_{k,n}(T)+1} - 1\right). \tag{26}$$

Taking the expectation of Eq. (26), applying Jensen's inequality on the result, and combining Eq. (3), we have

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{a_{k,n}(t)\} \geq \frac{1}{2}\left(\frac{1}{\mathbb{E}\{\frac{Y_{k,n}(T)}{T}\} + \frac{1}{T}} - 1\right). \tag{27}$$

Taking $T\to\infty$ and denoting $\hat{q}_{k,n}$ as the long-term throughput of $f_{k,n}$, i.e., $\hat{q}_{k,n} = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{y_{k,n}(t)\}$, we yield

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{a_{k,n}(t)\}$$
$$\geq \lim_{T\to\infty}\frac{1}{2}\left(\frac{1}{\mathbb{E}\{\frac{Y_{k,n}(T)}{T}\}+\frac{1}{T}}-1\right)$$
$$\geq \frac{1}{2}\left(\frac{1}{\hat{q}_{k,n}} - 1\right). \tag{28}$$

Combining Eq. (28) and the objective function in Eq. (9), the theorem can then be inferred. ■

***Theorem** 3:* Under Algorithm 1, the average long-term CA of all coflows approximately equals to two times the optimal objective value O^*_{P1} of problem **P1**.

Proof: Define O_{alg1} as the average long-term CA of all coflows achieved by Algorithm 1. Also, define O^*_{P2} as the optimal objective value of **P2**. Since Algorithm 1 is based on the optimal solution of **P2**, O_{alg1} can actually be characterized by O^*_{P2}. Therefore, we only need to compare O^*_{P2} to O^*_{P1}. Define $\hat{q}^L_{k,n}$ as the long-term throughput of $f_{k,n}$, under the optimal solution of problem **P3**. Consider a stationary randomized policy $R \in \Pi$, where the scheduling probabilities $\{\theta_{k,n}, \forall k, \forall n\}$ make the relevant long-term throughput $\hat{q}^R_{k,n} = p_{k,n}\theta_{k,n}$ equal to $\hat{q}^L_{k,n}$ for every $f_{k,n}$. In other words, R satisfies all throughput constraints, and hence is a feasible solution to problem **P2**. Let O^R_{P2} be the objective value of problem **P2**, under the policy R. Clearly, $O^R_{P2} \geq O^*_{P2}$. Combining Theorem 2, we have $\frac{O^*_{P2}}{O^*_{P1}} \leq \frac{O^R_{P2}}{O^*_{P3}}$. Comparing O^*_{P3}

with O_{P2}^R, we have $O_{P2}^R = 2 \times O_{P3}^* + 1$. This implies that $\frac{O_{P2}^*}{O_{P1}^*} \leq 2 + \frac{1}{O_{P3}^*}$. Thus, the theorem can be inferred. ■

The above theorem demonstrates that Algorithm 1 is nearly 2-competitive for the original problem **P1**, although it simply uses a sampling technique based on fixed scheduling probabilities calculated offline. In what follows, by taking advantage of Lyapunov optimization techniques [22], we develop another algorithm which is a little better than Algorithm 1 in terms of the optimality but requires additional information such as the current age of individual flows.

B. Drift-Plus-Age algorithm

The drift-plus-age algorithm focuses on decomposing the original problem **P1** into several sub-problems that can be sequentially solved in each time slot. In the design of this algorithm, we first transform the long-term throughput constraint in Eq. (5) into queue stability problem and then define a Lyapunov function to measure the aggregate congestion of all queues. The Lyapunov function has large value when the flows have less throughput than the required q_k. We proceed to define a Lyapunov drift to indicate the expected change in the Lyapunov function from one time slot to the next. Finally, at the beginning of each time slot, we minimize an upper bound on a drift-plus-age expression. The implication here is that the drift-plus-age algorithm is designed to reduce the average long-term CA while satisfying the throughput requirements of coflows by controlling the growth of the Lyapunov function.

Transforming throughput constraint: To accommodate the long-term throughput constraint in Eq. (5), we introduce a set of virtual queues $Q_{k,n}(t)$ for each flow $f_{k,n}$. Initially, each queue is empty at the beginning of the first time slot, i.e., $Q_{k,n}(0) = 0, \forall k, \forall n$. The queuing dynamics are as follows

$$Q_{k,n}(t+1) = \max\{Q_{k,n}(t) - y_{k,n}(t) + q_k, 0\}, \forall k, \forall n. \quad (29)$$

These queues take q_k as input and $y_{k,n}(t)$ as output. They essentially store the historical deviation from expected throughputs of coflows. The term q_k can be interpreted as the number of packets that should be delivered for flow $f_{k,n}$ during the time slot t. On the other hand, $y_{k,n}(t)$ can be viewed as the number of packets actually delivered during this time slot. As shown in Theorem 4, the constraint in Eq. (5) can exactly be characterized by these queues under the condition that the virtual queue is stable, i.e., $\lim_{T\to\infty} Q_{k,n}(T)/T{=}0$. Note that we will prove the strong stability of these virtual queues in Theorem 6 later.

Theorem 4: Eq. (29) is essentially equivalent to the long-term throughput constraint in Eq. (5), if the stability condition $\lim_{T\to\infty} Q_{k,n}(T)/T{=}0, \forall k, \forall n$ can be satisfied.

Proof: Considering the max function in Eq. (29), we have

$$Q_{k,n}(t+1) \geq Q_{k,n}(t) - y_{k,n}(t) + q_k. \quad (30)$$

Summing Eq. (30) over $t = 0, 1, \cdots, T-1$ and then dividing the result by T, we yield

$$\frac{Q_{k,n}(T) - Q_{k,n}(0)}{T} \geq -\frac{1}{T}\sum_{t=0}^{T-1} y_{k,n}(t) + q_k. \quad (31)$$

By taking expectations of both sides of the Eq. (31) and then substituting $Q_{k,n}(0) = 0$ and $\lim_{T\to\infty} Q_{k,n}(T)/T = 0$ into the result, we yield Eq. (5). Thus, the theorem is proved. ■

Characterizing drift-plus-age expression: We denote $\boldsymbol{Q}(t)$ as the concatenated vector of all queues, i.e., $\boldsymbol{Q}(t) = [Q_{1,1}(t), \ldots, Q_{1,N_1}(t), Q_{2,1}(t), \ldots, Q_{K,N_K}(t)]$. Then, we define the Lyapunov function as

$$L(\boldsymbol{Q}(t)){=}\frac{1}{2}\sum_{k=1}^{K}\sum_{n=1}^{N_k} Q_{k,n}(t)^2. \quad (32)$$

Such Lyapunov function quantitatively indicates the congestion of all queues. Moreover, it is large when queue backlogs are high, making more flows to have less throughput than the relevant required values. Obviously, the Lyapunov function needs to be persistently pushed towards a lower value, to keep the throughput constraint satisfied. To this end, we introduce the following Lyapunov drift to quantify the expected change in the Lyapunov function from one time slot to the next,

$$\Delta(\boldsymbol{Q}(t)) = \mathbb{E}\left\{L(\boldsymbol{Q}(t+1)) - L(\boldsymbol{Q}(t)) | \boldsymbol{Q}(t)\right\}. \quad (33)$$

As reducing $\Delta(\boldsymbol{Q}(t))$ in every time slot t only works for guaranteeing throughput requirements, we also need to consider the CA given by the objective function in problem **P1**. Therefore, we add up the drift and the CA—known as drift-plus-age, and then strive to minimize the drift-plus-age at the beginning of each time slot. In other words, we have the following sub-problem **P4** in each time slot t,

$$\min_{\{x_{k,n}(t)\}} \Delta(\boldsymbol{Q}(t)) + \frac{V}{K}\sum_{k=1}^{K} \max_{n=1,\ldots,N_k} \mathbb{E}\{a_{k,n}(t)\} \quad (34)$$
$$\text{s.t. Eqs. (1), (6), (7), (8),}$$

where V ($\geq$ 0) is a control parameter that depicts how much we shall emphasize the CA minimization, compared to guaranteeing throughput requirements.

Bounding drift-plus-age expression: Since directly minimizing the drift-plus-age expression in Eq. (34) relies on unknown information $Q_{k,n}(t+1)$, we seek to minimize its upper bound. To derive the upper bound of the drift-plus-age expression, we need the following theorem.

Theorem 5: In each time slot t, given any value of $\boldsymbol{Q}(t)$, the Lyapunov drift $\Delta(\boldsymbol{Q}(t))$ is bounded by

$$\Delta(\boldsymbol{Q}(t)) \leq H - \sum_{k=1}^{K}\sum_{n=1}^{N_k} \mathbb{E}\{Q_{k,n}(t)(y_{k,n}(t) - q_k)\}, \quad (35)$$

where $H = \frac{1}{2}\sum_{k=1}^{K}\sum_{n=1}^{N_k}(1 + q_k^2)$.

Proof: By taking advantage of the fact that for any $\rho, \varrho, \sigma \geq 0$, $(\max[\rho - \varrho + \sigma, 0])^2 \leq \rho^2 + \varrho^2 + \sigma^2 - 2\rho(\varrho - \sigma)$, we have the following inequality

$$\begin{aligned} Q_{k,n}^2(t+1) \leq & Q_{k,n}^2(t) + y_{k,n}^2(t) \\ & + q_k^2 - 2Q_{k,n}(t)(y_{k,n}(t) - q_k). \end{aligned} \quad (36)$$

Rearranging the terms in Eq. (36), and then combining the definition of $\Delta(\boldsymbol{Q}(t))$ and the fact that $y_{k,n}^2(t) \leq 1$, we yield Eq. (35). Thus, the theorem is proved. ■

Based on Theorem 5, we now can obtain the upper bound of the drift-plus-age expression in Eq. (34), by adding the expression $\frac{V}{K}\sum_{k=1}^{K} \max_{n=1,\ldots,N_k} \mathbb{E}\{a_{k,n}(t)\}$ to both sides of

Algorithm 2 Drift-Plus-Age algorithm

```
1: At the beginning of each time slot $t$, observe the current queue backlog $Q_{k,n}(t)$;
2: Determine the decision variables $x_{k,n}(t)$, $\forall k \in \mathcal{K}$, $\forall n \in \{1,\dots,N_k\}$ to minimize the objective function $\frac{V}{K}\sum_{k=1}^{K}\max_{n=1,\dots,N_k}\mathbb{E}\{a_{k,n}(t)\} - \sum_{k=1}^{K}\sum_{n=1}^{N_k}\mathbb{E}\{Q_{k,n}(t)(y_{k,n}(t)-q_k)\}$ in problem P5.
3: At the end of this time slot $t$, update $a_{k,n}(t)$ for each flow in each coflow, according to Eq. (2) and the actual value of the random variable $y_{k,n}(t)$.
```

Eq. (35). Till now, we have transformed the original problem **P1** to the following optimization **P5** at each time slot t

$$\begin{aligned}\min_{\{x_{k,n}(t)\}} \quad & \frac{V}{K}\sum_{k=1}^{K}\max_{n=1,\dots,N_k}\mathbb{E}\{a_{k,n}(t)\} \\ & -\sum_{k=1}^{K}\sum_{n=1}^{N_k}\mathbb{E}\{Q_{k,n}(t)(y_{k,n}(t)-q_k)\} \qquad (37)\\ \text{s.t. Eqs. } & (1),(6),(7),(8).\end{aligned}$$

The transformed problem **P5** is an integer programming with convex objective function. To solve it efficiently, one can use some commercial softwares, e.g., CPLEX [23] and Gurobi [24]. Yet, one can also develop efficient approximation algorithm. Due to page limit, we omit the details for brevity.

Algorithm 2 summarizes the whole procedure of the drift-plus-age algorithm. At the beginning of each time slot, based on current values of the queue backlog as well as the age of each flow, this algorithm strives to solve problem **P5**, so as to obtain the scheduling strategies $\{x_{k,n}(t)\}$. Then, at the end of this time slot t, it updates the age of each flow in each coflow, based on Eqs. (2) (3) and the actual value of the random variable $y_{k,n}(t)$. We now analyze the performance of Algorithm 2 through Theorem 6.

***Theorem** 6:* For any positive real value V, implementing Algorithm 2 for all time slots can achieve the following performance guarantee:

$$\frac{1}{K}\sum_{k=1}^{K}\bar{A}_k \le \frac{H}{V} + O^*_{P1}, \qquad (38)$$

$$\overline{Q} = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K}\sum_{n=1}^{N_k}\mathbb{E}\{Q_{k,n}(t)\} \le \frac{H+VO^*_{P1}}{\alpha}, \quad (39)$$

where O^*_{P1} denotes the optimal objective value of problem **P1**, $\alpha > 0$ and $\overline{Q}$ is the time-averaged queue length.

Proof: Consider a scheduling policy $R \in \Pi$ that chooses the scheduling decision $x_{k,n}(t)$, independent of the current queue backlogs, and yields the following steady values for all time slots $t = 0, 1, 2, \dots$:

$$\frac{1}{K}\sum_{k=1}^{K}\max_{n=1,\dots,N_k}\mathbb{E}\{a_{k,n}(t)\} = O^*_{P1}, \qquad (40)$$

$$\mathbb{E}\{y_{k,n}(t)\} \ge q_k + \alpha, \alpha \ge 0. \qquad (41)$$

As the existence of such scheduling policy can be proved by applying similar techniques in [22, 25], we omit the details for brevity. Algorithm 2 aims to select decisions that can minimize the upper bound of drift-plus-age expression, including the decisions in the above policy R. Thus, combining Eqs. (35), (40) and (41), we have

$$\begin{aligned}\Delta(\boldsymbol{Q}(t)) + \frac{V}{K}\sum_{k=1}^{K}\max_{n=1,\dots,N_k}\mathbb{E}\{a_{k,n}(t)\} \\ \le H + VO^*_{P1} - \alpha\sum_{k=1}^{K}\sum_{n=1}^{N_k}\mathbb{E}\{Q_{k,n}(t)\}.\end{aligned} \qquad (42)$$

Summing Eq. (42) over $t = 0, 1, \dots, T-1$ and dividing the result by T, we yield

$$\begin{aligned}&\frac{\mathbb{E}\{L(\boldsymbol{Q}(T))\}}{T} - \frac{\mathbb{E}\{L(\boldsymbol{Q}(0))\}}{T} + \\ &\frac{1}{K}\sum_{k=1}^{K}\max_{n=1,\dots,N_k}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{a_{k,n}(t)\} \\ \le \quad & H + VO^*_{P1} - \frac{\alpha}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K}\sum_{n=1}^{N_k}\mathbb{E}\{Q_{k,n}(t)\}.\end{aligned} \qquad (43)$$

Rearranging the terms of Eq. (43), using the fact that $L(\boldsymbol{Q}(t)) \ge 0$ and $a_{k,n}(t) \ge 0$, we yield

$$\begin{aligned}&\frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=1}^{K}\sum_{n=1}^{N_k}\mathbb{E}\{Q_{k,n}(t)\} \\ &\qquad \le \frac{H + VO^*_{P1} + \mathbb{E}\{L(\boldsymbol{Q}(0))\}/T}{\alpha}.\end{aligned} \qquad (44)$$

By taking $T\to\infty$, we can yield Eq. (39). Similarly, rearranging the terms of Eq. (43), we get

$$\begin{aligned}&\frac{1}{K}\sum_{k=1}^{K}\max_{n=1,\dots,N_k}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{a_{k,n}(t)\} \\ &\qquad \le \frac{H}{V} + O^*_{P1} + \frac{\mathbb{E}\{L(\boldsymbol{Q}(0))\}}{VT}.\end{aligned} \qquad (45)$$

Taking $T\to\infty$ again and using Eq. (4), we yield Eq. (38). ■

Theorem 6 demonstrates that by applying Algorithm 2 with an arbitrarily larger V, we can make the average long-term CA of all coflows arbitrarily close to the optimum O^*_{P1} while satisfying the throughput requirements as virtual queues are stable according to Eq. (39).

IV. Performance Evaluation

In this section, we evaluate our proposed algorithms via large-scale simulations based on a real-world data trace.

Comparing methods: We compare the following two methods with the proposed algorithms throughout our simulations:

- **MinAF**: The MinAF (Minimum-coflow-Age-First) prioritizes all coflows according to their CAs, and schedules a coflow with minimum CA each time.
- **LASF (Aalo)**: All coflows are prioritized and scheduled based on their *current* coflow sizes—how many packets a coflow has already been transferred. We refer to this method as Least-Attained-Service-First (LASF), and it is conceptually equivalent to Aalo's scheduling policy [7].

To ease the presentation, we denote the proposed two algorithms as **SR** (Stationary Randomized) and **DPA** (Drift-plus-Age), respectively.

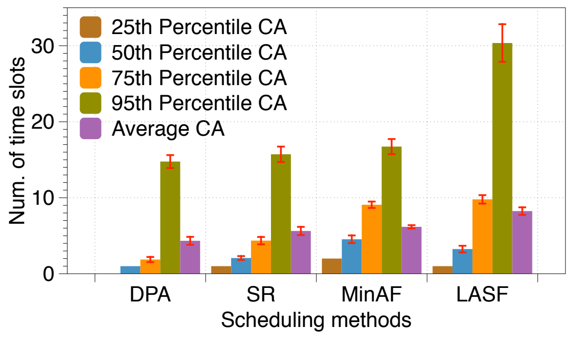


Fig. 2. The 25th, 50th, 75th, 95th percentile and average CA across all coflows and all time slots.

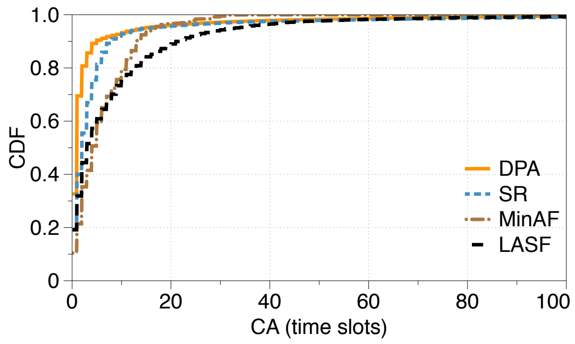


Fig. 3. CDF of CA across all coflows and all time slots.

A. Methodology

Simulator: As the transmission unit is packet, we develop a packet-level simulator to evaluate the proposed algorithms. We choose the network size based on Facebook data center fabric [6, 7] and simulate a network with 150 servers (i.e., $M = 150$). In our simulation, the length of each time slot is taken as the time to transmit a maximum-sized packet (e.g., 1500-byte) [16], i.e., 10μs. In this context, the incoming/outgoing link of each server is uniformly set to be 1.2Gbps.

Workload: We conduct the simulations based on a data trace collected on a 3000-machine 150-rack cluster of Facebook [6, 7]. This data trace contains 526 coflows that are scaled down to the rack-level, making it exactly match the simulated data center network. Recall that in this paper, a coflow refers to a set of flows that constantly transmit packets from a fixed set of sources to another fixed set of destinations. Thus, we use a subset of coflows (i.e., K=100) and make them long-running to coexist in the network. We keep the original senders/receivers for each coflow, but randomly generate packets for each sender-receiver pair with a fixed probability (i.e., $p_{k,n}$). Each $p_{k,n}$ is randomly selected in the range of $(0, 1)$. The throughput requirement for the flows in each coflow k, i.e., q_k, is set to $\min_n p_{k,n}/K$.

Metrics: We evaluate all the methods with the following six metrics by default: the 25th, 50th, 75th, 95th and average CA of all coflows, and the number of unsatisfied coflows. As aforementioned, the CA of a coflow is the maximum age of all its flows, which is measured in time slots. An unsatisfied coflow refers to a coflow that is unsatisfied with its throughput requirement. We run each method 100 tries. Each try lasts for a total of 500 time slots, with random workload as is aforementioned. Note that DPA is simulated with $V = 9.5 \times 10^4$.

B. Performance of the proposed algorithms

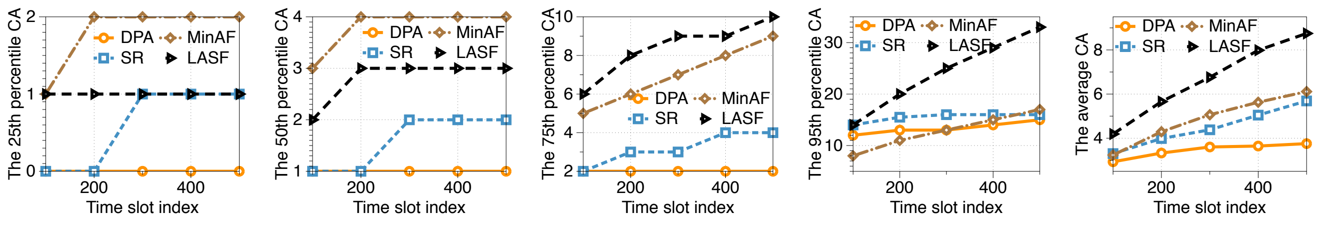
CA minimization: We first demonstrate the performance merits of our proposed algorithms in minimizing the CA of coflows. Fig. 2 depicts the 25th, 50th, 75th, 95th and average CA across all coflows and all time slots, for DPA, SR, MinAF, and LASF. Each bar value in this figure is an average of 100 tries. The error bar represents the standard deviation of the relevant measurement. We can observe that among the four methods, DPA is the best while SR is the second-best, with respect to the 25/50/75/95th and average CA. Specifically, compared to the methods of MinAF and LASF, our DPA can reduce the 25th/50th/75th/95th/average CA of all coflows across all time slots by 100%/77.9%/79.4%/12.0%/29.8% and 100%/69.2%/80.9%/51.6%/47.5%, respectively. Moreover, for our SR algorithm, such CA reductions over MinAF and LASF can also be 50%/54.4%/52.0%/6%/9% and 0%/36.3%/49.3%/48.2%/31.8%, respectively. These results directly demonstrate the efficiency of our proposed algorithms in reducing the CA of coflows. From Fig. 2, we can further observe that our DPA has a much stabler performance than the other methods, in terms of the 25th, 50th, 75th, and 95th percentile CA. For instance, the 95th percentile CA of DPA varies from 13 to 17, with standard deviation being 0.85; for LASF, it varies from 26 to 37, and the standard deviation is 2.48. As for the average CA, our DPA algorithm achieves a little bit lower stability performance than MinAF and LASF. However, the highest value of the average CA of the 100-try achieved by DAP is even smaller than the lowest value of both MinAF and LASF. SR's stability performance is not as prominent as DPA does. The underlying reason is that except for the random process in the workload, SR involves one more random process than DPA, i.e., scheduling coflows according to precomputed probabilities.

TABLE I
AVG. NUMBER OF UNSATISFIED COFLOWS OF 100 TRIES.

Scheduling methods	DPA	SR	MinAF	LASF
Number of unsatisfied coflows	0 ± 0	0 ± 0	1.77 ± 1.09	0 ± 0

Throughput requirement guarantee: We now investigate the performance of our proposed algorithms in guaranteeing the throughput requirements of coflows. As shown in Table I, all the methods, excluding MinAF, can satisfy the throughput requirements of all coflows. More specifically, for MinAF, the number of unsatisfied coflows can vary from 0 to 5, with the average number of unsatisfied coflows and the standard deviation being 1.77 and 1.09, respectively. The reason is that MinAF makes low-age coflows to be scheduled repeatedly, leaving high-age coflows to transmit packets less often and hence resulting in unsatisfied coflows.

Deep dive analysis: To understand the underlying reasons for the improvements of our proposed algorithms, we first plot the CDFs of the CA across all coflows and all time slots during one specific try of each method in Fig. 3. The results from



(a) 25th (b) 50th (c) 75th (d) 95th (e) Average

Fig. 4. The 25th, 50th, 75th, 95th percentile and average CA of coflows over time.

Fig. 3 are in line with that from Fig. 2, namely, both DPA and SR outperform the other two methods. To be more specific, most of the CA values, i.e., 90%, are less than 5 (time slots) for DPA; the corresponding fractions for SR, MinAF, and LASF are 81.3%, 59.2%, and 60.1%, respectively. Combining Fig. 2 and Fig. 3, we observe that LASF performs worse than MinAF in minimizing CA. This is because that LASF prefers coflows with small current sizes, while such coflows often have a large coflow width (i.e., number of flows) and account for a small portion of the entire data-trace. As such, more server ports will be occupied, leaving fewer scheduling opportunities for the majority of coflows in the trace.

We further plot the 25/50/75/95/average CA of coflows over time for different methods in Fig. 4. We can clearly observe from this figure that most of the curves rise as time goes by. This is reasonable because the CA of each coflow will continually grow if it has not been served. We further observe that most of the time, DPA performs better than other methods. One may question why MinAF achieves the lowest 95th percentile CA before the time slot 300. The reason is that the low-age coflows scheduled by MinAF may not transmit any packet given the randomness in the workload, thus making high-age coflows to have some probabilities to be scheduled in the future. Despite this, when time goes by long enough, the 95th percentile CA of MinAF will increase beyond those of our DPA and SR algorithms.

V. Related Work

In this section, we only review the closely related work along either coflow scheduling or age optimization.

Coflow scheduling: Coflow scheduling has gained significant research attention in recent years. However, all existing efforts on coflow scheduling rely on the CCT metric to achieve different design goals including fairness [11, 12], guaranteeing deadlines [6, 26] and minimizing CCTs [6–10, 27]. Nevertheless, the CCT is a one-shot performance metric, and it cannot reflect the staleness of the output results of the streaming application. Hence, we present a new metric named CA, which is capable of continuously measuring how long the coflow destinations have not received new packets and accordingly can satisfy the needs of streaming applications better.

AoI optimization: Despite the growing literature in AoI optimization (e.g., [13, 14, 28]), some intrinsic differences prevent us from directly adopting them for solving our CA minimization problem. First, they are only applicable in wireless networks, where the wireless channels are unreliable. In contrast, the links in the data center network are relatively stable. Second, they care more about the freshness of the input data, while we only concern how long the output results have not been updated. Finally, they do not take into account the application-level semantics and thus are coflow-agnostic.

VI. Conclusions

In this paper, we propose a new performance metric, namely CA, to quantify the transmissions of coflows generated by distributed streaming applications. The metric CA tracks the longest time-since-last-service among all flows in a coflow, and it is more aligned with the essential needs of streaming applications on the freshness of their output results. With coflows generated by streaming applications running on a shared data center, we study the problem of minimizing the average long-term CA while guaranteeing the throughput requirements of coflows. We develop a rigorous mathematical model and formulate a long-term stochastic optimization problem which is challenging to solve due to its inherent complexity and the uncertainties in packet arrivals. To solve this problem efficiently, we have designed two efficient algorithms: stationary randomized algorithm (SR) and drift-plus-age algorithm (DPA). We have validated the performance of each algorithm via rigorous theoretical analysis and large-scale simulations. Theoretical results have shown that SR is nearly 2-competitive for the original problem while DPA is near-to-optimal. Simulation results further demonstrate that DPA is better than SR, but both of them can outperform the state-of-the-art method in minimizing the CA and incur no violation on the throughput requirements of coflows.

Acknowledgment

This work was supported partly by the NSFC General Technology Basic Research Joint Funds under Grant U1836214; the State Key Program of National Natural Science of China under Grants 61832013; the Artificial Intelligence Science and Technology Major Project of Tianjin under Grant 18ZXZNGX00190; the National Key R&D Program of China under Grant 2019QY1302; the NSFC under Grant 61672379; the National Key R&D Program of China under Grant 2019YFB2102404; the NSFC-Guangdong Joint Funds under Grant U1701263; the Natural Science Foundation of Tianjin under Grant 18ZXZNGX00040; the National Key R&D Program of China under Grant 2018YFB1004700; the NSFC under Grants 61872265, 61672131; the Key research and Development Program for Guangdong Province 2019B010136001; the NSFC under Grants 61772112, U1811463; the Science Innovation Foundation of Dalian under Grant 2019J12GX037. Wenyu Qu and Heng Qi are the corresponding authors: wenyu.qu@tju.edu.cn, hengqi@dlut.edu.cn.

References

[1] W. Lin, H. Fan, Z. Qian, J. Xu, S. Yang, J. Zhou, and L. Zhou, "Streamscope: Continuous reliable distributed processing of big data streams." in *Proc. of USENIX NSDI*, 2016.

[2] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance." in *Proc. of USENIX NSDI*, 2017.

[3] S. Rajadurai, J. Bosboom, W.-F. Wong, and S. Amarasinghe, "Gloss: Seamless live reconfiguration and reoptimization of stream programs," in *Proc. of ACM ASPLOS*, 2018.

[4] B. Heintz, A. Chandra, and R. K. Sitaraman, "Trading timeliness and accuracy in geo-distributed streaming analytics," in *Proc. of ACM Symposium on Cloud Computing (SoCC)*, 2016.

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

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

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

[8] 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.

[9] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. Lau, "Efficient online coflow routing and scheduling," in *Proc. of ACM MobiHoc*, 2016.

[10] W. Li, X. Yuan, K. Li, Q. Heng, and X. Zhou, "Leveraging endpoint flexibility when scheduling coflows across geo-distributed datacenters," in *Proc. of IEEE INFOCOM*, 2018.

[11] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max-min fairness," in *Proc. of IEEE INFOCOM*, 2016.

[12] W. Wang, S. Ma, B. Li, and B. Li, "Coflex: Navigating the fairness-efficiency tradeoff for coflow scheduling," in *Proc. of IEEE INFOCOM*, 2017.

[13] I. Kadota, A. Sinha, and E. Modiano, "Scheduling algorithms for optimizing age of information in wireless networks with throughput constraints," in *Proc. of IEEE INFOCOM*, 2018.

[14] Q. He, D. Yuan, and A. Ephremides, "Optimal link scheduling for age minimization in wireless systems," *IEEE Transactions on Information Theory*, vol. 64, no. 7, pp. 5381–5394, 2018.

[15] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and degradation in jetstream: Streaming analytics in the wide area." in *Proc. of USENIX NSDI*, 2014.

[16] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 307–318, 2015.

[17] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching." in *Proc. of USENIX NSDI*, 2017.

[18] W. Li, D. Niu, Y. Liu, S. Liu, and B. Li, "Wide-area spark streaming: Automated routing and batch sizing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1434–1448, 2018.

[19] R. G. Gallager, *Stochastic processes: theory for applications*. Cambridge University Press, 2013.

[20] "Cvx research." [Online]. Available: http://cvxr.com/

[21] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[22] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[23] "Cplex optimizer." [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[24] "Gurobi optimizer." [Online]. Available: http://www.gurobi.com/

[25] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data centers power reduction: A two time scale approach for delay tolerant workloads," in *Proc. of IEEE INFOCOM*, 2012.

[26] R. Xu, W. Li, K. Li, and X. Zhou, "Shaping deadline coflows to accelerate non-deadline coflows," in *Proc. of IEEE/ACM IWQoS*, 2018.

[27] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: near-optimal network design for coflows," in *Proc. of ACM SIGCOMM*, 2018.

[28] Y. Sun, E. Uysal-Biyikoglu, R. D. Yates, C. E. Koksal, and N. B. Shroff, "Update or wait: How to keep your data fresh," *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7492–7508, 2017.