Efficient Online Coflow Routing and Scheduling

Yupeng Li[1,2]† Shaofeng H.-C. Jiang[1]† Haisheng Tan[2]* Chenzi Zhang[1]†
Guihai Chen[3] Jipeng Zhou[2] Francis C.M. Lau[1]
[1] the University of Hong Kong (HKU), Pokfulam, Hong Kong
[2] Jinan University (JNU), Guangzhou, China
[3] Shanghai Jiao Tong University (SJTU), Shanghai, China

ABSTRACT

A *coflow* is a collection of related parallel flows that occur typically between two stages of a multi-stage compute task in a network, such as shuffle flows in MapReduce. The coflow abstraction allows applications to convey their semantics to the network so that application-level requirements (e.g., minimizing the completion time of the slowest flow) can be better satisfied. In this paper, we study the routing and scheduling of multiple coflows to minimize the average coflow completion time (CCT). We first propose a rounding-based randomized approximation algorithm, called `OneCoflow`, for single coflow routing and scheduling. The multiple coflow problem is more challenging as coexisting coflows will compete for the same network resources such as link bandwidths. To minimize the average CCT, we derive an online multiple coflow routing and scheduling algorithm, called `OMCoflow`, and prove that it has a reasonably good competitive ratio. To the best of our knowledge, this is the first online algorithm with theoretical performance guarantees which considers routing and scheduling simultaneously for multi-coflows. Compared with existing methods, `OMCoflow` runs more efficiently, and it avoids the problem of frequently rerouting the flows. Extensive simulations on a Facebook data trace show that `OMCoflow` outperforms the state-of-the-art heuristic schemes significantly (e.g., reducing the average CCT by up to 41.8% and the execution time by up to 99.2% against RAPIER [28]).

CCS Concepts

•Networks → Network protocol design; Network resources allocation; •Theory of computation → Network optimization;

*Contact H. Tan at hstan.hku@gmail.com.

†Part of the work was done when they were visiting at JNU.

MobiHoc'16, July 04-08, 2016, Paderborn, Germany
© 2016 ACM. ISBN 978-1-4503-4184-4/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2942358.2942367

Keywords

Coflow; Routing and scheduling; Online algorithm; Data Center Networks

1. INTRODUCTION

Distributed computing frameworks such as MapReduce [14], Dryad [22] and Spark [26] are very popular among cloud applications. In these frameworks, data flows for one job may share a common performance goal, such as minimizing the completion time of the slowest flow. However, such application-level requirements are often largely overlooked when cloud providers aim at optimizing network-level metrics such as the (individual) flow completion time.

The *coflow* abstraction was first proposed in [10] to bridge the above gap, which is defined as a group of parallel flows that are related and that come about typically between two stages of a job (i.e. shuffle flows in MapReduce). To improve application-level performance, other than individual flows, coflows should be considered such that job-specific requirements could be better satisfied. Take MapReduce tasks as an example. Generally, a reduce job cannot start before its shuffle flows are all completed. Thus, we can treat these correlated flows as a coflow, and try to minimize the maximum completion time among these flows, which is called the *coflow completion time* (CCT). Furthermore, in many situations, multiple coflows could coexist in a network, i.e., the newly coming ones and the existing ones that have not finished their transmission. Therefore, we take *the average coflow completion time* (average CCT) as a performance metric in this paper, similar to [10, 11, 12, 13, 15, 24, 27, 28]. As coexisting coflows will compete for the communication resources (e.g., routing paths and link bandwidth), multiple coflow scheduling and routing is challenging.

Most previous works on network-level optimization were agnostic on the existence of coflows (e.g., [5, 8, 7, 19]), which actually could harm the application-level performance (examples can be found in [10, 13, 11]). Here, we focus on coflow-aware results; we list the state-of-the-art research outputs in Table 1.

Some papers [10, 11, 12, 13, 24, 27, 9, 23] studied coflow scheduling only. Varys [13] proposed effective heuristics to schedule coflows aiming at minimizing the average CCT and meeting coflow deadlines. Qiu et al. [24] proposed the first deterministic algorithm with a constant approximation ratio for multiple coflow scheduling. Their algorithm was based on an offline model which is not so practical, which assumes the information of all the coflows was given at the beginning. Chen et al. [9] designed a new utility optimal scheduler for coflows, which defined different levels of sensitivity regarding completion time for different coflows. Without prior knowledge of coflows, Aalo [11] adopted Discretized Coflow-Aware Least-Attained Service (D-CLAS) to separate coflows into priority queues based on the amounts they have already sent. Dogar et al. [15] and Luo et al. [23] investigated the decentralized coflow-aware scheduling problem. CODA [27] was the first work to recognize coflows among individual flows using machine learning techniques. An error-tolerant coflow scheduler was then proposed and implemented.

Without considering flow routing, scheduling-only coflow transmission approaches may fail to minimize the CCT (such examples can be found in [28]). RAPIER [28] was the first publication (and the only one so far to the best of our knowledge) that considered coflow routing and scheduling simultaneously. However, their solution for multiple coflows was heuristics-based and hence lack theoretical performance guarantees on minimizing the average CCT. The multiple coflow routing and scheduling problem is challenging due to: 1) routing and scheduling the flows of a single coflow to minimize the CCT is NP-hard [16]; 2) more than one coflow in the network might concurrently compete for the transmission resources (such as the link capacity); and 3) in most practical scenarios imaginable, we do not have the information about the future coflows, and hence an online solution is called for. In this paper, we provide new solutions to the multi-coflow routing and scheduling problem. We start with single coflow routing and scheduling, and then propose an efficient online multiple coflow routing and scheduling scheme with a theoretical performance guarantee. Specifically, we have the following contributions:

- For single coflow routing and scheduling, we propose a randomized algorithm, called `OneCoflow`, to minimize the CCT with an approximation ratio of $4\ln 2n$ with high probability, where n is the number of nodes in the network (Sec. 3).
- We derive an online algorithm, called `OMCoflow`, to solve the multiple coflow routing and scheduling problem (Sec. 4). We show that `OMCoflow` has a good *competitive ratio* in minimizing the average CCT of the coflows. To the best of our knowledge, this is the first result for online multi-coflow routing and scheduling with theoretical performance guarantees.
- Compared with existing heuristic approaches (e.g., RAPIER [28]), `OMCoflow` can run much more efficiently. More importantly, `OMCoflow` routes each flow in the coflows only once while RAPIER would frequently reroute all the flows in the network when a new coflow arrives or an existing coflow finishes transmission.
- We conducted extensive simulations using a real-world data trace collected from Facebook (also adopted in [11, 13]). Compared to RAPIER, `OMCoflow` reduces the average CCTs by 41.8% (11.7% at the 95th percentile), and dramatically reduces the algorithm's execution time by up to 99.2% (Sec. 5).

Organization: The rest of the paper is organized as follows. We first present the system model and problem formulation in Sec. 2. In Sec. 3, we study the single coflow routing and scheduling, and derive a randomized approximation algorithm called `OneCoflow`. We propose an online algorithm called `OMCoflow` for the multiple coflow routing and scheduling problem in Sec. 4. Extensive simulations are in Sec. 5. Finally, we conclude this work and point out some future work in Sec. 6.

2. MODEL AND PROBLEM DEFINITIONS

2.1 System Model

A network is modeled as a directed graph $G = (V, E)$, where E is the edge set and V is the node set. The network size is denoted as $n = |V|$. In data center networks (DCNs), each node $v \in V$ can be a server or a switch. Each edge $e \in E$ has capacity R_e.

A *coflow* is a collection of related parallel flows with a common performance goal (e.g. to minimize the maximum flow completion time). Denote C_i $(1 \le i \le m)$ as the coflow arriving at time T_i, which contains ω_i individual flows. A *flow* j $(1 \le j \le \omega_i)$ within a coflow C_i is defined by a 3-tuple $(s_j, t_j, v_j) \in C_i$, where $s_j, t_j \in V$ are the source and destination nodes, and $v_j > 0$ is the flow volume. The available path set for flow j in coflow C_i is denoted as $P_j^{(i)}$. Generally, the number of paths for a pair of nodes in the network can be as large as $O(n!)$. However, for networks with specific structures (e.g. Fat tree [3] and VL2 [18]), the data flow between two nodes would follow some given paths. Therefore, in this paper, we assume the number of paths between any pair of nodes is bounded by $K = \text{poly}(n)$.

Without loss of generality, we assume that a coflow C_i has all the information about its flows and starts transmission as soon as it arrives at the network at time T_i, similar to [10, 5, 12, 15, 13, 28]. At time $x \ge T_i$, a flow $(s_j, t_j, v_j) \in C_i$ is then forced to be routed on a path $p_j^{(i)}(x) \in P_j^{(i)}$ with a rate $b_j^{(i)}(x)$, which is in fact the bandwidth assigned to this flow at that time. Note that $b_j^{(i)}(x)$ can be zero for some time x's, which means this flow is *waiting* for transmission. Therefore, a *routing and scheduling strategy* for a coflow C_i is defined as[1]

$$S_i := \{p_j^{(i)}(x), b_j^{(i)}(x)\}_{j=1}^{\omega_i}.$$

A time-slotted system is considered. We then define *the coflow completion time* (CCT) F_i for coflow C_i to be the minimum time such that

$$\sum_{x=T_i}^{T_i+F_i} b_j^{(i)}(x) \ge v_j^{(i)}, \quad \textit{for all} \quad 1 \le j \le \omega_i,$$

which means the earliest time that all the flows in C_i finish transmitting their data. We further define the total CCT for all the coflows as $F = \sum_{i=1}^m F_i$.

Since frequent flow reroutings will lead to severe coordination overhead, which is not desirable in practice, therefore, in our model, each coflow (and therefore each of its flows) is allowed to be routed only once[2]. We say $\{S_i\}_{i=1}^m$ is a *valid* strategy, if for all $x \ge 0$, for all $e \in E$,

$$\sum_{i=1}^{m}\sum_{j=1}^{\omega_i} I(e \in p_j^{(i)}) \cdot b_j^{(i)}(x) \le R_e.$$

[1]In this paper, we denote a set $\{a_1, a_2, \ldots, a_n\}$ as $\{a_i\}_{i=1}^n$.
[2]Therefore, each flow will be transmitted along exactly one path, while the bandwidth assigned may be changed from time to time.

Table 1: Comparison of Research Outputs Related to Coflows

Solution	Scheduling	Routing	Coflow-aware	Online	Performance Guarantee
Hedera [4], D^3 [25], PDQ [20], etc.	✓	✗	✗	✓	✗
pFabric [5], etc.	✓	✗	✗	✓	✓
Orchestra [12], Varys [13], Baraat [15], Luo et al. [23], Aalo [11], CODA [27]	✓	✗	✓	✓	✗
Qiu et al. [24], Chen et al. [9],	✓	✗	✓	✗	✓
RAPIER [28]	✓	✓	✓	✓	✗
`OMCoflow` [this work]	✓	✓	✓	✓	✓

Here, $I(\chi) = 1$ if event χ is true and $I(\chi) = 0$ otherwise. That is to say, for a valid strategy, at any time the sum of the bandwidth requirements on any link cannot exceed its link capacity.

2.2 Problem Definition

With the above settings, we define the *Online Multiple Coflow Routing and Scheduling Problem* as follows.

PROBLEM 1. *In a network, m coflows $C_1, C_2, \ldots, C_m$ arrive at time $T_1, T_2, \ldots, T_m$. The information of every coflow $C_i := \{(s_j, t_j, v_j)\}_{j=1}^{\omega_i}$ is given at arrival including the corresponding source-destination pair, the volume, and the available paths $P_j^{(i)}$ of each of the ω_i individual flows. The problem is to design an algorithm to find a valid routing and scheduling strategy $\{S_i\}_{i=1}^m$ for each coflow so that the average completion time of the coflows, $\frac{F}{m}$, is minimized.*

In our multi-coflow routing and scheduling problem, information about future coflows is not known. Therefore, an *online* algorithm is desired. In the following, we start with investigating a special case where there is only one single coflow in the network (i.e., $m = 1$). Then, based on the single coflow scheduling and routing solution, we propose an online algorithm to tackle the problem of multiple coflows.

3. SINGLE COFLOW ROUTING AND SCHEDULING

In this section, we consider the special case where there is only one coflow $C := \{(s_j, t_j, v_j)\}_{j=1}^{\omega}$ in the network. According to the above, for each flow $j \in C$, the source-destination pair, the volume, and the available paths P_j are given. The capacity for each edge e is R_e. Our goal is to find a valid strategy for C with the minimum coflow completion time (CCT).

3.1 Approximation Algorithm

In the following, we derive a randomized algorithm for the single coflow routing and scheduling problem, which runs in polynomial time to the number of nodes n and returns a valid strategy with CCT that is at most $4 \ln 2n$ times the optimal solution with high probability.

The main technique used in our algorithm is convex programming and rounding. We first use Program P to capture the problem. But as we shall see later, Program P is not convex and not efficiently solvable. We resolve the issue by using a binary search based approach to tackle Program P efficiently. However, even we have the solution for Program P, it does not give a routing and scheduling strategy directly. Therefore, we need another procedure, which is usually called *rounding*, to get a valid strategy.

The whole algorithm that combines all these ingredients is given in Algorithm 1, named as `OneCoflow`. We introduce the details by first describing Program P:

$$
\begin{aligned}
&\text{minimize} \quad t && \text{(P)}\\
&\text{subject to} \quad \sum_{j=1}^{\omega} b_j \sum_{p \in P_j} I(e \in p) \cdot x_{j,p} \le R_e, \quad e \in E, && \text{(P.a)}\\
&x_{j,p} = 0, \quad \text{if } \exists e \in p, b_j > R_e, && \text{(P.b)}\\
&\sum_{p \in P_j} x_{j,p} = 1, \quad 1 \le j \le \omega, && \text{(P.c)}\\
&0 \le x_{j,p} \le 1, \quad p \in P_j, 1 \le j \le \omega, && \text{(P.d)}\\
&b_j t = v_j, \quad 1 \le j \le \omega. && \text{(P.e)}
\end{aligned}
$$

In this program, variable t denotes the CCT of C. We use variable $x_{j,p}$ to denote whether we choose path p for the j-th flow, which is intended to be an integer variable in {0,1} but relaxed to be a real number. Variable b_j denotes the *average* bandwidth of the j-th flow, which is defined by the last constraint (P.e). The constraint (P.a) captures the requirement of a valid strategy. Furthermore, we include the constraint (P.b) in the program in order to reduce the integrality gap and the rounding algorithm can benefit from it. Constraint (P.c) is intended to ensure each flow get routed along exactly one path. The fact that Program P is a relaxation of our problem will be fromally proved in Lemma 1.

We then give `OneCoflow` (Algorithm 1) to solve the single coflow routing and scheduling with the minimum CCT. `OneCoflow` starts with the optimal solution of Program P

Algorithm 1: OneCoflow: Single Coflow Strategy

```
  Input: Coflow $C := \{(s_j, t_j, v_j)\}_{j=1}^{\omega}$, $\{P_j\}_{j=1}^{\omega}$
  Output: A valid strategy for $C$
1 Let $(b, x, t)$ be an optimal solution of Program P on $C$.
2 for $j = 1, 2, \ldots, \omega$ do
3     Set $p_j = p'$ with probability $x_{j,p'}$, for $p' \in P_j$.
4 Let $\alpha \ge 1$ be the smallest real number such that
  $\{p_j, \frac{b_j}{\alpha}\}_{j=1}^{\omega}$ is a valid strategy.
5 Return $\{p_j, \frac{b_j}{\alpha}\}_{j=1}^{\omega}$.
```

(Line 1). Then in the *for* loop, `OneCoflow` finds paths for each flow in C independently. For each j, we sample a path p' for it with probability $x_{j,p'}$. In the end, we rescale the bandwidth to make sure the strategy is valid (Line 4). Note that in our model (Sec. 2.1), the bandwidth is a function of time, but `OneCoflow` returns a constant value (Line 5). This implies that, for flow j, the bandwidth function is equal to

$\frac{b_j}{\alpha}$ when transmitting the corresponding flow, and is equal to zero after the completion of the corresponding flow.

3.2 Analysis

As stated above, in some important scenarios such as data center networks, we assume that the number of paths between any pair of nodes is bounded by $K = \text{poly}(n)$.

THEOREM 1. *For any coflow $C := \{(s_j, t_j, v_j)\}_{j=1}^{\omega}$ that is defined on a directed graph G with n vertices such that $\max_{j:1\le j\le\omega} |P_j| = K_C$, `OneCoflow` (Algorithm 1) runs in $O(LP(O(\omega K_C), O(\omega K_C + n^2)))$ and gives a valid strategy with CCT at most $4\ln 2n$ times the optimal CCT of C with probability at least $\frac{3}{4}$, where $LP(x, y)$ denotes the time complexity of solving a linear program with x variables and y constraints*[3]*.*

PROOF. Our proof contains three parts: 1) we first give a lower bound of the minimum CCT of C in Lemma 1; 2) then we give an upper bound of the CCT of the strategy returned by our algorithm `OneCoflow` in Lemma 2; and 3) we address the time complexity in the end.

1) Lower Bound of the Optimal CCT: We will prove that P is a relaxation of our single coflow routing and scheduling problem. Let OPT_P denote the optimal objective of P, and let OPT denote the minimum completion time of C. The fact of relaxation implies that $\text{OPT}_\text{P} \le \text{OPT}$.

LEMMA 1 (P IS A RELAXATION). *For any valid strategy $S := \{(p_j, b_j)\}_{j=1}^{\omega}$ for C, there exists a feasible solution of P whose objective value in P is the same as the completion time of S.*

PROOF. Suppose T is the completion time of S. We define a solution of P as follows.

- Set $t = T$.
- For $1 \le j \le \omega$, set $x_{j,p_j} = 1$, and set $x_{j,p'} = 0$ for all p' such that $p' \in P_j$ and $p' \ne p_j$.
- For $1 \le j \le \omega$, set $b_j = \frac{v_j}{T}$.

Observe that it only remains to verify that constraint (P.a)

$$\sum_{j=1}^{\omega} b_j \sum_{p\in P_j} I(e \in P) \cdot x_{j,p} \le R_e,$$

for $e \in E$. By the definition of the completion time, we have $\sum_0^T b_j(x) \ge v_j$. This implies that

$$b_j = \frac{v_j}{T} \le \frac{\sum_0^T b_j(x)}{T}.$$

On the other hand, by the fact that S is a valid strategy and by the definition of validness, for all e, we know that

$$\sum_{i=1}^{\omega} I(e \in p_j) \sum_0^T b_j(x) \le TR_e.$$

Therefore, we get

$$\sum_{j=1}^{\omega} b_j \sum_{p\in P_j} I(e \in p) \cdot x_{j,p} \le \frac{\sum_{j=1}^{\omega} I(e \in p_j) \sum_0^T b_j(x)}{T} \le R_e.$$

This implies the Lemma. □

[3]A linear program is polynomial solvable, and is very fast in most cases. In addition, here we can boost the success probability of our algorithm by running it multiple times.

2) Upper Bound of our CCT: The solution for P is not directly a strategy, and we need to construct one from it. This procedure is usually called *rounding*. Moreover, since the rounding is randomized, it is important to guarantee that our algorithm performs well with a high probability.

LEMMA 2. *Algorithm 1 returns a strategy with completion time at most $4\ln 2n$ times the minimum completion time of C with probability at least $\frac{3}{4}$.*

PROOF. For $e \in E$, define

$$\alpha_e := \frac{\sum_{j=1}^{\omega} I(e \in p_j) \cdot b_j}{R_e}.$$

Then, $\alpha = \max_e \alpha_e$. It is sufficient to prove that

$$\Pr[\alpha > 4\ln 2n] \le \sum_{e\in E} \Pr[\alpha_e > 4\ln 2n] \le \frac{1}{4n^2} \cdot n^2 = \frac{1}{4}$$

by the union bound, and this completes the proof.

Next, we fix e and focus on the proof of $\Pr[\alpha_e > 4\ln 2n] \le \frac{1}{4n^2}$. Define random variable $X_j := I(e \in p_j) \cdot b_j$, and $X := \sum_{j=1}^{\omega} X_j$. Note that we have the following properties.

- All X_j's are independent.
- X_j is either 0 or b_j, and
$$\Pr[X_j = b_j] = \sum_{p\in P_j} I(e \in p) \cdot x_{j,p}.$$
- $E[X] = \sum_{j=1}^{\omega} b_j \cdot \sum_{p\in P_j} I(e \in p) \cdot x_{j,p} \le R_e$, by the constraint (P.a) of Program P.
- If $\Pr[X_j = b_j] > 0$ then $b_j \le R_e$. This is also by the constraint (P.b) of Program P.

We can then apply Lemma 3 on the subset of random variables $\{X_j : 1 \le j \le \omega, \Pr[X_j = b_j] > 0\}$, and we get

$$\Pr[\alpha_e > 4\ln 2n] \le \exp(-2\ln 2n) = \frac{1}{4n^2},$$

which completes our proof. □

LEMMA 3. *Suppose $\{X_i\}_{i=1}^{\omega}$ are independent random variables, such that X_i $(1 \le i \le \omega)$ takes value 0 with probability $1 - p_i$ $(0 < p_i < 1)$, and takes value v_i with probability p_i. Define $X := \sum_{i=1}^{\omega} X_i$. Then for $R \ge 0, \beta \ge 2e$ such that $v_i \le R$ and $E[X] \le R$, we have $\Pr[X > \beta R] \le \exp(-\frac{\beta}{2})$.*

PROOF. Let $t > 0$ be a parameter to be fixed. Then

$$\begin{aligned}\Pr[X > \beta R] &= \Pr[tX > t\beta R] \\ &= \Pr[\exp(tX) > \exp(t\beta R)] \\ &\le \exp(-t\beta R) \cdot E[\exp(tX)] \\ &= \exp(-t\beta R) \prod_{i=1}^{\omega} E[\exp(tX_i)],\end{aligned}$$

where the inequality is by the Markov's inequality.

We analyze $E[\exp(tX_i)]$. By definition, we derive

$$\begin{aligned}E[\exp(tX_i)] &= (1 - p_i) + p_i \exp(tv_i) \\ &= 1 + p_i(\exp(tv_i) - 1) \\ &\le \exp(p_i(\exp(tv_i) - 1)),\end{aligned}$$

where the last inequality is from the fact that $1+x \le \exp(x)$ for all real x. Then, we get

$$\prod_{i=1}^{\omega} E[\exp(tX_i)] \le \exp(\sum_{i=1}^{\omega} p_i(\exp(tv_i)-1)).$$

We pick t to satisfy $\exp(tv_i) \le 1 + \frac{1}{2}t\beta v_i$, for all i.

We will show the existence and give the value of such a t later. With this property of t, we have

$$\prod_{i=1}^{\omega} E[\exp(tX_i)] \le \exp(\sum_{i=1}^{\omega} \frac{1}{2}t\beta p_i v_i) \le \exp(\frac{1}{2}t\beta R),$$

where the last inequality is by $E[X] \le R$.

Therefore, we get

$$\Pr[X > \beta R] \le \exp(-t\beta R + \frac{1}{2}t\beta R) = \exp(t(-\frac{1}{2}\beta R)).$$

We define $t := \frac{\ln \frac{\beta}{2}}{R}$. Recall that we require t to satisfy the property that $\exp(tv_i) \le 1 + \frac{1}{2}t\beta v_i$ for all i. To verify the property,

$$\begin{aligned}\exp(tv_i) &= \exp(\frac{v_i}{R}\ln\frac{\beta}{2}) = (\frac{\beta}{2})^{\frac{v_i}{R}} \\ &\le 1 + \frac{\beta}{2}\cdot\frac{v_i}{R} \le 1 + \frac{1}{2}t\beta v_i,\end{aligned}$$

where the second last inequality follows from the fact that $a^x \le 1 + ax$, for $a \ge 1$, $0 \le x \le 1$. (To verify the fact, we let $g(x) = a^x - 1 - ax$. Then $g'(x) = \ln a \cdot a^x - a$, and we know that $g(x)$ takes the maximum when $x = 0$ or $x = 1$, and therefore $g(x) \le \max\{g(0), g(1)\} = 0$).

Substituting t, we have

$$\Pr[X > \beta R] \le \exp(-\frac{\beta}{2}\ln\frac{\beta}{2}) \le \exp(-\frac{\beta}{2}),$$

which completes our proof. $\square$

Therefore, the strategy returned by the rounding scheme has completion time at most $O(\log n)\mathrm{OPT_P}$, with high probability. Together with $\mathrm{OPT_P} \le \mathrm{OPT}$, we immediately get the desired approximation ratio.

3) Time Complexity of OneCoflow: Program P is not convex, because of the constraints P.b and P.e. Hence, it is non-trivial to find the optimal solution in polynomial time.

To resolve the issue, we propose *a binary search based approach*. The main observation is that if t is fixed instead of a variable, the constraints are all linear. This means that we can solve a feasibility linear program to test whether a specific t is feasible or not.

Hence, our algorithm for solving Program P is as follows. First we binary search t. Then for a fixed t, we define $b_j = \frac{v_j}{t}$ (note that b_j is constant given t), and we solve the following feasibility linear program FLP:

$$\begin{aligned}
&\text{minimize} \quad 0 && \text{(FLP)}\\
&\text{subject to} \quad \sum_{j=1}^{\omega} b_j \sum_{p\in P_j} I(e\in p)\cdot x_{j,p} \le R_e,\ e\in E, && \text{(FLP.a)}\\
&\quad x_{j,p} = 0, \qquad \text{if } \exists e\in p, b_j > R_e, && \text{(FLP.b)}\\
&\quad \sum_{p\in P_j} x_{j,p} = 1, \qquad 1\le j\le\omega, && \text{(FLP.c)}\\
&\quad 0 \le x_{j,p} \le 1, \qquad p\in P_j, 1\le j\le\omega. && \text{(FLP.d)}
\end{aligned}$$

When the linear program is infeasible, we enlarge t; otherwise, we search for smaller t. In the end, we will find the nearly optimal t (we do not need the exact t since we are seeking for approximation algorithms anyway), and the corresponding solution for Program P is constructed by solving Program FLP with the binary search based approach.

Therefore, during solving Program P, we compute O(1) Program FLPs with $O(\omega K_C)$ number of variables, and $O(\omega K_C + n^2)$ number of constraints. The number of feasibility linear programs that we need to solve should typically be a constant in reality. After we get the optimal solution, the main algorithm, i.e. Algorithm 1, runs in time $O(\omega K_C)$. In conclusion, our algorithm `OneCoflow` runs in $O(\mathrm{LP}(O(\omega K_C), O(\omega K_C + n^2)))$.

This ends the proof of Theorem 1. $\square$

Extension to General Networks In data center networks, the number of paths between any pair of nodes is bounded by $K = \mathrm{poly}(n)$. So our algorithm `OneCoflow` is able to compute a valid strategy for a coflow in polynomial time. But the assumption $K = \mathrm{poly}(n)$ may be not applicable for other networks, for example, wireless networks with an arbitrary number of nodes. In such a network, the number of paths between any pair of nodes can be super polynomial, which would make the algorithm fail to solve Program P in polynomial time. Specifically, we consider the case when P_j for (s_j, t_j) are defined as *all* the possible directed simple paths from s_j to t_j using edges in some $E_j \subseteq E$. In this setting, we have Theorem 2.

THEOREM 2. *For any coflow $C := \{(s_j, t_j, v_j)\}_{j=1}^{\omega}$ with P_j defined to be all the (simple directed) paths from s_j to t_j using edge set $E_j \subseteq E$, there exists an algorithm that returns a valid strategy for C with completion time at most $(4\ln 2n)$ times the minimum completion time of C with probability at least $\frac{3}{4}$, running in time $O(LP(O(\omega|E|), O(\omega|E|)))$, where $LP(x, y)$ means the running time of solving a linear program with x variables and y constraints.*

PROOF. The proof framework is similar to that for `OneCoflow`. However, since the way we define the path set changes, we have to modify the program a little. Specifically, we use variables $x_{j,e}$ to denote whether edge e is chosen in the j-th flow, instead of variables $x_{j,p}$ to denote the selection of paths in Program P. The constraints of the program are changed accordingly. We also adopt the binary search based approach to solve the new program. After solving the program optimally, we design a different rounding procedure to get a valid routing. Finally, as in `OneCoflow`, we scale down a factor of the bandwidth to make sure the strategy is valid. The analysis of the approximation ratio is mostly similar to that before, and the key ingredient is the use of our measure concentration result (Lemma 3). Due to the space, we omit the details here. $\square$

4. ONLINE MULTI-COFLOW ROUTING AND SCHEDULING

In this section, we study online multiple coflow routing and scheduling, where m coflows $\{C_i\}_{i=1}^m$ arrive at time $\{T_i\}_{i=1}^m$[4]. Without loss of generality, we assume $T_1 = 0$. At

[4]We allow multiple coflows arrive at the same time, i.e., if C_i and C_{i+1} arrive at t simultaneously, we set $T_i = T_{i+1} = t$.

T_i, it is required to give the coflow $C_i := \{(s_j^{(i)}, t_j^{(i)}, v_j^{(i)})\}_{j=1}^{\omega_i}$ a strategy $S_i := \{(p_j^{(i)}, b_j^{(i)}(x))\}_{j=1}^{\omega_i}$ for $x \geq T_i$.

Our aim is to design an algorithm to allocate the network resources through computing for each unfinished coflow a valid strategy. The algorithm design targets at the following properties: 1) performance guarantee: The algorithm is competitive with a non-trivial ratio for the online problem; 2) practical for real application: we do not allow frequent reroutings; specifically, recall that in our model, one coflow (and therefore one flow) will be routed only once; and 3) fairness: those coflows with a larger completion time will be provided relatively more network resources.

4.1 Online Algorithm

We make use of `OneCoflow`, our algorithm for the single coflow strategy, as a black box, and design a competitive algorithm for the online multiple coflow routing and scheduling problem. The main idea is that when a new coflow arrives or a coflow completes its transmission, to compute each existing coflow a valid strategy and to fully utilize the link capacity, we reschedule each flow's bandwidth by scaling down its bandwidth computed by `OneCoflow`. The algorithm is called `OMCoflow`, as shown in Algorithm 2.

Algorithm 2: OMCoflow: Online Algorithm for Multiple Coflow Routing and Scheduling

```
  Input: $\{(C_i, T_i)\}_{i=1}^m$
  Output: Valid strategies for $\{(C_i, T_i)\}_{i=1}^m$
1 while there is a new coflow coming or some existing coflow being completed do
2     Suppose $I$ is the set of indices of coflows that are not completed at this point.
3     for $i \in I$ do
4         Define $\alpha_i := \frac{\sqrt{\mathrm{OPT}_i}}{\sum_{l \in I} \sqrt{\mathrm{OPT}_l}}$, where $\mathrm{OPT}_i$ is the optimal of Program P for coflow $C_i$,
5         Update $S_i := \{(p_j^{(i)}, \alpha_i b_j^{(i)})\}_{j=1}^{\omega_i}$, where $\{(p_j^{(i)}, b_j^{(i)})\}_{j=1}^{\omega_i}$ is the solution computed by OneCoflow (with inputs as $C_i$ and $\{P_j^{(i)}\}_{j=1}^{\omega_i}$) when $C_i$ arrived.
6     Scale the bandwidths for all flows in $\{S_i\}_{i \in I}$ by the same factor that is the largest to make the strategy still valid.
```

`OMCoflow` computes a valid strategy for each existing coflow in the network including the newly arrived one. A new coflow arriving or some coflow being completed would wake up `OMCoflow` (Line 1). It calls `OneCoflow` for each coflow C_i to compute a valid strategy when C_i arrives at the network assuming that C_i monopolizes the network (i.e., the case of single coflow routing and scheduling). The output of `OneCoflow` for C_i, $\{(p_j^{(i)}, b_j^{(i)})\}_{j=1}^{\omega_i}$, is then stored to avoid duplicated computations in the future. Note that the interplay of concurrent coflows is likely to make this strategy invalid, so we scale down each coflow's bandwidth with a factor (Line 4 and Line 5). The factor α_i is special to each coflow, and is computed with respect to the optimal of Program P for all concurrent coflows. So the new scheduling is a weighted combination of the single coflow scheduling, and α_i's are the weights. In this way, obviously, each coflow receives a valid strategy. In particular, for coflow C_i, we base on the ρ-approximate optimal single coflow routing and scheduling (by `OneCoflow`), and scale down the bandwidth of the flows uniformly by α_i. As we always make use of the valid strategy output by `OneCoflow` (through only scale the coflow bandwidth), `OneCoflow` is applied only for the newly arrived coflow, and we will solve Program P only once for each coflow. For the case when a coflow is completed, we do not need to call `OneCoflow`. Moreover, after we reweight the scheduling, we scale all the flow bandwidths by the same largest possible factor, to make use of the rest of the bandwidth (Line 6).

4.2 Analysis

Preliminaries: When we say an algorithm for the problem is ρ-competitive, we mean for all k $(1 \leq k \leq m)$, the online algorithm returns a valid strategy for each coflow in the set $\{(C_i, T_i)\}_{i=1}^k$ that has a completion time at most ρ times its minimum completion time under the offline setting. Offline setting means the information of the posterior coflows is known in advance (the knowledge of (C_j, T_j) for $j > i$).

Theorem 3 shows that `OMCoflow` has a provable competitive guarantee. Although the analysis is not tight, experiments in Sec. 5 show that our algorithm actually has superb performance.

THEOREM 3. *Suppose the competitive ratio of `OneCoflow` (Algorithm 1) is ρ. Then, `OMCoflow` (Algorithm 2) is $m\rho$-competitive for the online multiple coflow routing and scheduling problem, running in time $O(m) \cdot TIME_{SIN}$, where $TIME_{SIN}$ is the time complexity of `OneCoflow`.*

PROOF. Since the time complexity part is immediate, we focus on the analysis of the competitive ratio.

Lower Bound of Offline Optimal Suppose OPT is the offline minimum completion time for $\{(C_i, T_i)\}_{i=1}^m$. Let OPT_i be the optimal value of the Program P for coflow C_i. It is immediate that each coflow C_i contributes to the total completion time with no less than its minimum completion time when it monopolizes the network. Moreover, since OPT_i is a lower bound of the minimum completion time of coflow C_i when it is the only coflow that is running, we conclude that $\mathrm{OPT} \geq \sum_{i=1}^m \mathrm{OPT}_i$.

Upper Bound the Competitive Ratio Since we already have the lower bound $\mathrm{OPT} \geq \sum_{i=1}^m \mathrm{OPT}_i$, we compare the performance of our algorithm to $\sum_{i=1}^m \mathrm{OPT}_i$.

LEMMA 4. *$ALG \leq m\rho \sum_{i=1}^m OPT_i$, where ALG denotes the completion time of the scheduling given by Algorithm 2.*

PROOF. Consider the optimization problem for some subset I of $\{1, 2, \ldots, m\}$,

$$\begin{aligned} \text{Minimize} \quad & \sum_{i \in I} \frac{\mathrm{OPT}_i}{x_i} \\ \text{subject to} \quad & \sum_{i \in I} x_i \leq 1, \end{aligned}$$

where variables x_i's are non-negative. Applying the Cauchy-Schwarz inequality,

$$\sum_{i \in I} \frac{\mathrm{OPT}}{x_i} \geq (\sum_{i \in I} \frac{\mathrm{OPT}_i}{x_i}) \cdot (\sum_{i \in I} x_i) \geq (\sum_{i \in I} \sqrt{\mathrm{OPT}_i})^2.$$

Moreover, the minimizer is $x_i = \frac{\sqrt{\mathrm{OPT}_i}}{\sum_{l \in I} \sqrt{\mathrm{OPT}_l}}$.

Thus, for each I in each iteration of the algorithm, the factors α_i's ($i \in I$) are optimally picked with respect to I.

Define $\alpha_i^{(m)} := \frac{\sqrt{\text{OPT}_i}}{\sum_{l=1}^m \sqrt{\text{OPT}_l}}$. Define ALG_i to be the completion time of the single coflow scheduling returned by the ρ-approximate algorithm for coflow C_i. We observe that for any integer $i \in I$,

$$\alpha_i \geq \frac{\sqrt{\text{OPT}_i}}{\sum_{l=1}^m \sqrt{\text{OPT}_l}} = \alpha_i^{(m)}.$$

Hence, the completion time F_i for each coflow C_i is at most

$$\frac{\text{ALG}_i}{\alpha_i^{(m)}} \leq \rho \frac{\text{OPT}_i}{\alpha_i^{(m)}} \leq \rho \sqrt{\text{OPT}_i} \sum_{l=1}^m \sqrt{\text{OPT}_l}.$$

Hence, we have

$$\text{ALG} \leq \sum_{i=1}^m \frac{\text{ALG}_i}{\alpha_i^{(m)}} \leq \rho (\sum_{i=1}^m \sqrt{\text{OPT}_i})^2 \leq m\rho \cdot \sum_{i=1}^m \text{OPT}_i.$$

The last inequality is by the Cauchy-Schwarz inequality. □

Therefore, our theorem is proved. □

We can combine the results of Theorem 1 and Theorem 3 to get the following corollary.

Corollary 1. *OMCoflow is $(4m \ln 2n)$-competitive with high probability, where m is the number of coflows and n is the number of nodes in the network.*

4.3 Advantages of OMCoflow

OMCoflow computes a valid strategy for each unfinished coflow with theoretical performance guarantees. Besides, OMCoflow has the following significant merits.

No Frequent Rerouting: OMCoflow computes the routes for each flow in a coflow only once when the coflow arrives at the network. That is to say, each flow will transmit its data exactly along one path, although the bandwidth assigned to this path might change from time to time. Avoiding frequent rerouting will make OMCoflow more efficient and more implementable in practical applications.

Fairness: We observe that the weight factors α_i are defined to be proportional to $\sqrt{\text{OPT}_i}$. This ensures that those coflows with large completion time when they monopolize the network will be given more bandwidth, and those coflows with small completion time will be given relatively less bandwidth. This shows that our algorithm inherently has the guarantee of fairness. Moreover, as there is barely any sign of waiting in any coflow, our algorithm also prevents *starvation* permanently.

Work Conserving: Line 6 in Algorithm 2 corresponds to a speed-up operation which scales up the flow bandwidths with a factor to completely utilize the link capacity. This illustrates the good property of work conservation of our OMCoflow algorithm.

5. PERFORMANCE EVALUATION

We evaluate our proposed online algorithm OMCoflow by large-scale simulations based on real-application data from Facebook [1], which is also used by authors [11, 13]. We develop a trace-driven simulator which compares OMCoflow with the following three state-of-the-art schemes: 1) **Baseline:** all the individual flows are routed by ECMP [21], in which the bandwidths are assigned following *the max-min fairness*; 2) **Heuristic:** routing and scheduling coflows according to the minimum remaining time first (MRTF) heuristic, as in RAPIER [28]; and 3) **Scheduling-only:** scheduling coflows using the MRTF heuristic and routes all the individual flows by ECMP, as in Varys [13]. Here we highlight our key results:

Table 2: Coflows categorized by length (Short and Long) and width (Narrow and Wide) in our original Facebook trace.

Coflow Type	SN	LN	SW	LW
Length	Short	Long	Short	Long
Width	Narrow	Narrow	Wide	Wide
% of Coflows	59.51%	16.53%	11.79%	12.17%
% of Total Bytes	0.01%	0.11%	0.87%	99.01%

- For both Facebook and Fat-tree topologies, OMCoflow outperform the Baseline, Heuristic and Scheduling-only schemes, by reducing the average (95th percentile) C-CT by up to 76.7% (62.4%), 41.8% (11.7%) and 65.0% (46.1%), respectively (Sec. 5.2.1).

- OMCoflow has much shorter average CCTs when applied to specific categories of coflows with different widths and sizes (Sec. 5.2.1).

- OMCoflow runs much faster than Heuristic, the state-of-the-art algorithm considering routing and scheduling simultaneously (Sec. 5.2.2). It dramatically reduces the execution time by up to 99.2%, which makes OMCoflow more practical in real applications.

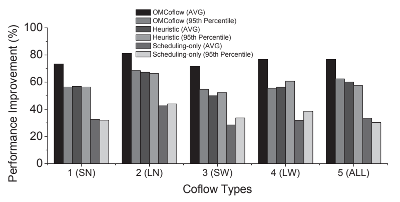
- OMCoflow consistently outperforms the other schemes over a wide range of values for different coflow parameters, such as the number of coflows, the size, the width and the intercoflow arrival intervals (Sec. 5.3).

5.1 Methodology

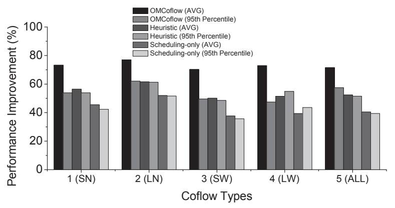
Data Trace Our workload is based on a Hive/MapReduce trace that was collected on a 3000-machine 150-rack cluster with 10:1 oversubscription ratio, which was also adopted in [13]. The synthesized trace contains 526 coflows that are scaled down to the rack-level with exact inter-arrival times. Mappers in the same rack are combined into one rack-level mapper, and so are the reducers. Communication patterns at rack-level are captured accurately and the amounts of data (e.g., the coflow size) being shuffled are rounded to the nearest megabyte [1]. Since the original trace only has the whole data size of a coflow shuffled to the reducers, we uniformly sample the size for each flow within it.

Coflow A coflow is measured by 3 key parameters:1) *width:* the total number of individual flows; 2) *length:* the size of its largest flow it contains; and 3) *size:* the total amount of data in megabytes. Similar to [13, 24], we divide non-zero coflows into 4 categories as shown in Table 2. A coflow is W *(wide)* if it involves more than 50 flows, and otherwise it is N *(narrow)*; and a coflow is L *(long)* if its length is greater than 5MB, and otherwise it is S *(short)*.

Simulator Existing packet-level simulators (e.g., ns-2) are not suitable here due to their high communication overheads. Similar to previous works (e.g., [4, 11, 13, 24, 28]), we develop a trace-driven simulator with the embedded solver



(a) Facebook



(b) Fat-tree

Figure 1: Improvements in the average and 95th percentile CCTs of `OMCoflow`, Heuristic and Scheduling-only compared with Baseline in (a) Facebook topology and (b) Fat-tree topology, respectively.

GUROBI [2] to solve the linear programming problem. Our simulator runs on a machine with two Intel Xeon CPU E5-2690 v2 3.00GHz with 32GB memory and 1TB hard drive.

We mainly choose the Facebook data center fabric [6, 17] as our network topology, where there are 15 pods each with 10 racks and 4 fabric switches. In one pod, each rack is connected to each of the 4 fabric switches. The capacity of a link connecting fabric switches with racks is 1Gbps. Each fabric switch is connected to different groups of 5 upper-layer spine switches, so we have 4×5=20 spine switches in total. The capacity of each link between a fabric and the spine layer is 4Gbps. Moreover, we investigate a 10-ary Fat-tree [3] fabric in our experiments as well. In this topology, we fix the first 150 racks as the end points of the coflows. The link capacity between the racks and edge switches is set to 1Gbps, so are the links between the edges and the aggregation layers. Links between the top two layers have a capacity of 4Gbps.

Metrics We measure the improvement in the average CCT when comparing two schemes. Take comparing Scheme X with the Baseline as an example:

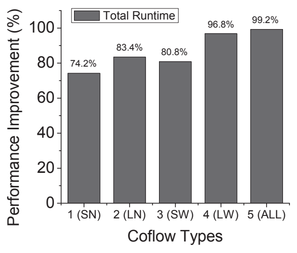
$$\text{Improvement } (\%) = \frac{\text{average CCT(Baseline)} - \text{average CCT(X)}}{\text{average CCT(Baseline)}}.$$

We measure the CCTs at high percentile (the 95th percentile) similarly.

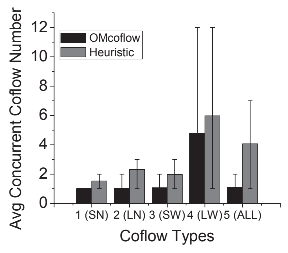
5.2 OMCoflow Performance

5.2.1 *CCTs in Different Network Topologies*

In this section, we evaluate the coflow completion times (CCTs) of `OMCoflow`, Heuristic and Scheduling-only against the Baseline in the Facebook topology and Fat-tree topology respectively. We not only investigate the whole data trace in [13] of totally 526 coflows, but also the subsets of coflows in different categories defined in Table 2 based on the lengths and widths of the coflows. Figure 1 illustrates our results. Figure 1(a) shows that with the Facebook topology, when we consider the whole data trace, `OMCoflow`, Heuristic and Scheduling-only improve the average (95th percentile) CCT by up to 76.7% (62.4%), 60.0% (57.4%) and 33.5% (30.2%), respectively. That is to say, `OMCoflow` relatively improves the average (95th percentile) CCT by 41.8%(11.7%) over Heuristic and 65.0% (46.1%) over Scheduling-only. We can also see that the performance of `OMCoflow` is steady for special categories of coflows. Figure 1(b) shows a similar result in the Fat-tree topology.



(a) Improvements in the running time



(b) the average (maximum and minimum) number of concurrent coflows

Figure 2: Improvements in the runtime using `OMCoflow` w.r.t. Heuristic in the Facebook topology.

5.2.2 *Running Time*

We compare the time efficiency of `OMCoflow` with Heuristic, the state-of-the-art heuristic scheme (such as RAPIER in [28]) that also takes into account routing and scheduling simultaneously. Here we use the Facebook topology. The metric we consider is given as

$$\text{Improvement } (\%) = \frac{\text{Runtime(Heuristic)} - \text{Runtime}(\texttt{OMCoflow})}{\text{Runtime(Heuristic)}}.$$

In Figure 2(a), we can see that `OMCoflow` dramatically outperforms Heuristic for all different coflow categories. For the whole data trace, it reduces the running time by up to 99.2% (equivalently 135× faster). We analyze the reason as follows. `OMCoflow` and Heuristic are both executed when a new coflow arrives or an existing coflow completes transmission. Note that each of the LPs to be solved in Heuristic or `OMCoflow` have the same numbers of variables and constraints (see Program (4) in [28] and Program FLP in Sec. 3), so the time complexity to solve an LP in both schemes is the same.Next, we investigate the numbers of LPs that each scheme need solve under the following two cases.

- When a new coflow arrives, Heuristic needs to solve $O(\Delta_m^2)$ LPs, which can reach $O(m^2)$ in the worst cases. Here, Δ_m is the number of concurrent coflows in the network [28] and m is the total number of coflows. In contrast, `OMCoflow` only needs to solve $O(1)$ LPs as analyzed in Sec. 3.2.
- When an existing coflow completes transmission and leaves, Heuristic still has to solve $O(\Delta_m^2)$ LPs, while `OMCoflow` need not solve any LP but only scale the flow bandwidth. (Lines 4–6 in Algorithm 2).

We further investigate the number of concurrent coflows when executing the schemes. Figure 2(b) illustrates that `OMCoflow` has less concurrent coflows on average. For both schemes, solving LPs accounts for most of the runtime.

Therefore, `OMCoflow` can be executed much faster than Heuristic and is more practical to real applications.

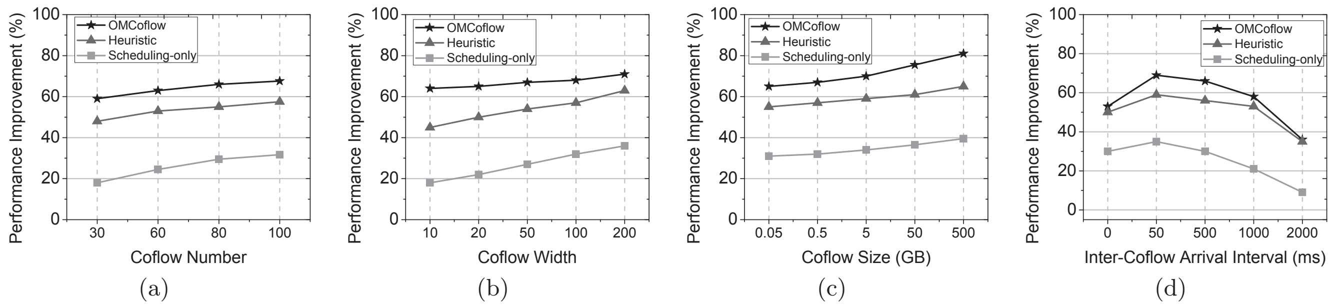


Figure 3: The impact of parameters on the average CCTs of `OMCoflow`, Heuristic and Scheduling-only compared to Baseline.

5.3 Impact of Coflow Parameters

In this section, we study the impacts of key coflow parameters on the performance of `OMCoflow`, such as the total coflow number, the coflow width, the coflow size, and the inter-coflow arrival interval. Unless otherwise specified, we assume the Facebook data center topology. Generally, in Figure 3, we can see that `OMCoflow` always outperforms Baseline, Heuristic and Scheduling-only under different scenarios.

5.3.1 Coflow Number

To evaluate the impact of the coflow number on the performance of the routing and scheduling schemes, we fix the other parameters, i.e., setting the coflow width, the coflow size and the mean inter-coflow arrival interval as 100, 500MB and 100ms, respectively. We then inject different numbers of coflows into the network, and calculate the improvement of the average CCTs for OMCoflow, Heuristic, and Scheduling-only compared with Baseline.

Figure 3(a) shows that the improvement in average CCT increases with the growth of the coflow numbers using each of the three schemes. The reason is that more coflows would lead to a more severe competition for the network resources, and then the efficient solutions will gain more benefits. `OMCoflow` can reduce the average CCT by up to 67.6% (see the case of 100 coflows) when compared with the Baseline, which outperforms the other two schemes.

5.3.2 Coflow Width

Recall that the coflow width is the number of flows within it. In this experiment, we fix the coflow number, the size and and the mean inter-coflow arrival interval as 100, 500MB and 100ms, respectively, and then study the influence of the coflow width to the average CCTs.

Figure 3(b) shows that the larger coflow width, the more improvement in average CCT gained by each of the three schemes. The reason is still that more data communications lead to more severe competition for the network resources. We can observe that `OMCoflow` always outperforms Heuristic and Scheduling-only. `OMCoflow` can reduce the average CCT by up to 70.7% compared to the Baseline, while Heuristic and Scheduling-only can only reduce it by 62.5% and 36.3%.

5.3.3 Coflow Size

Here we fix the coflow number, the width and the mean inter-coflow arrival interval to be 100, 100 and 100ms respectively. In each experiment, we then send coflows with the same size into the network.

For different coflow sizes, Figure 3(c) shows `OMCoflow` can reduce the average CCT by up to 80.6% when compared with the Baseline. When compared with Heuristic and Scheduling-only, OMCoflow can reduce the time by 44.7% and 67.6 %, respectively.

In addition, for all the three schemes, their improvement in the average CCTs increases with the growth of coflow sizes, while the improvement of `OMCoflow` escalates more dramatically than the other two. This is because under the online setting, the larger coflow size leads to more severe collisions among coflows. Compared with Heuristic, `OMCoflow` can result in a much smaller number of concurrent coflows in the network. Compared with Scheduling-only, `OMCoflow` also conducts routing, which contributes to the relief of the collisions. Therefore, the larger the coflow size is, the more improvements `OMCoflow` gains.

5.3.4 Inter-Coflow Arrival Interval

For each of the experiments in this part, the mean inter-coflow arrival interval is fixed to a value by modifying the intervals in the original data trace. The other parameters are fixed as the previous sections. We investigate the intervals starting from 0, which means all coflows arrive at the same time (the same as the offline model).

The results are shown in Figure 3(d). We can see that `OMCoflow` always outperforms Heuristic and Scheduling-only when the intervals are not extremely large. `OMCoflow` can reduce the average CCT by up to 69.0% when compared with Baseline. Moreover, we can observe when the interval becomes extremely large, the improvements of the schemes will decrease significantly, i.e., when the interval is 2s, little improvement can be gained. The reason is that if the interval is too large, most coflows will finish transmission during the interval and there will be little interaction among the coflows. In this case, the coflow routing will contribute more than the coflow scheduling to minimize the CCTs. Since both Heuristic and `OMCoflow` consider coflow routing, they have a better perfomance than Scheduling-only. Moreover, we can see that `OMCoflow` performs even better than Heuristic, which illustrates the advantage of our single coflow routing and scheduling algorithm, `OneCoflow`, when compared to the single coflow algorithm in RAPIER.

6. CONCLUSION

This paper studies the combined problem of routing and scheduling of coflows. Coflow is a relatively new concept which was proposed to bridge the semantic gap between the application-level requirements and the network-level metric-

s. For the single coflow routing and scheduling, we propose OneCoflow, a randomized algorithm to minimize the coflow completion time (CCT), which has a good approximation ratio with high probability. We further derive an online algorithm, called OMCoflow, to tackle the multiple coflow routing and scheduling problem. Our theoretical analysis shows that our algorithm has a good competitive ratio in minimizing the average CCT of the coflows. Extensive simulations based on a real-world data trace from Facebook indicated that OMCoflow outperforms the state-of-the-art multi-coflow routing and scheduling approaches in minimizing the average CCT and reducing the execution time. Consequently, OMCoflow is more suitable for practical applications. Existing works on coflow-aware routing and scheduling all assume that the flows in a coflow arrive at the network simultaneously, or equivalently, they make the coflow start after all of its flows have arrived. However, in real scenarios, flows within a coflow come more naturally to the network at different time slots, and the ones arriving earlier should not need to wait for the other flows. We will take this into account in our future work. OMCoflow is actually a randomized approach. Therefore, it is also meaningful and challenging to derive a deterministic online multi-coflow scheduling and routing scheme with good theoretical performance guarantees.

Acknowledgments

This work is supported in part by NSFC Grants 61502201, 61472252, 61373125, China 973 project (2014CB340303), NSF-Guangdong Grant 2014A030310172, and Hong Kong RGC CRF Grant C7036-15G.

7. REFERENCES

[1] Facebook Hive/MapReduce Trace. https://github.com/coflow/coflow-benchmark.

[2] GUROBI. http://www.gurobi.com/.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *Proc. of ACM SIGCOMM*, 2008.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. of USENIX NSDI*, 2010.

[5] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal Near-Optimal Datacenter Transport. In *Proc. of ACM SIGCOMM*, 2013.

[6] A. Andreyev. Introducing Data Center Fabric, the Next-Generation Facebook Data Center Network. https://code.facebook.com/posts/360346274145943.

[7] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. Information-Agnostic Flow Scheduling for Commodity Data Centers. In *USENIX NSDI*, 2015.

[8] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *Proc. of ACM CoNEXT*, 2011.

[9] L. Chen, W. Cui, B. Li, and B. Li. Optimizing Coflow Completion Times with Utility Max-Min Fairness. In *Proc. of IEEE INFOCOM*, 2016.

[10] M. Chowdhury and I. Stoica. Coflow: A Networking Abstraction for Cluster Applications. In *Proc. of ACM HotNets*, 2012.

[11] M. Chowdhury and I. Stoica. Efficient Coflow Scheduling without Prior Knowledge. In *Proc. of ACM SIGCOMM*, 2015.

[12] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing Data Transfers in Computer Clusters with Orchestra. In *Proc. of ACM SIGCOMM*, 2011.

[13] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient Coflow Scheduling with Varys. In *Proc. of ACM SIGCOMM*, 2014.

[14] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[15] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized Task-Aware Scheduling for Data Center Networks. In *Proc. of ACM SIGCOMM*, 2014.

[16] S. Even, A. Itai, and A. Shamir. On the Complexity of Time Table and Multi-Commodity Flow Problems. In *Proc. of IEEE FOCS*, 1975.

[17] N. Farrington and A. Andreyev. Facebook's Data Center Network Architecture. In *Proc. of IEEE Optical Interconnects*, 2013.

[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 *Proc. of ACM SIGCOMM*, 2009.

[19] H. Han, S. Shakkottai, C. Hollot, R. Srikant, and D. Towsley. Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Eploit Path Diversity in the Internet. *IEEE/ACM Transactions on Networking*, 14(6):1260–1271, 2006.

[20] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing Flows Quickly with Preemptive Scheduling. In *Proc. of ACM SIGCOMM*, 2012.

[21] C. E. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. In *RFC 2992*, 2000.

[22] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *ACM Eurosys*, 2007.

[23] S. Luo, H. Yu, Y. Zhao, B. Wu, S. Wang, and L. Li. Minimizing average coflow completion time with decentralized scheduling. In *Proc. of IEEE ICC*, 2015.

[24] Z. Qiu, C. Stein, and Y. Zhong. Minimizing the Total Weighted Completion Time of Coflows in Datacenter Networks. In *Proc. of ACM SPAA*, 2015.

[25] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better Never Than Late: Meeting Deadlines in Datacenter Networks. In *Proc. of ACM SIGCOMM*, 2011.

[26] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *USENIX HotCloud*, 2010.

[27] 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.

[28] Y. Zhao, K. Chen, W. Bai, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang. RAPIER: Integrating Routing and Scheduling for Coflow-Aware Data Center Networks. In *IEEE INFOCOM*, 2015.