Non-preemptive Coflow Scheduling and Routing

Ruozhou Yu, Guoliang Xue, Xiang Zhang, Jian Tang

***Abstract*—As more and more data-intensive applications have been moved to the cloud, the cloud network has become the new performance bottleneck for cloud applications. To boost application performance, the concept of coflow has been proposed to bring application-awareness into the cloud network. A coflow consists of many individual data flows, and a coflow is completed only when all its component flows are transmitted. The network performance of a cloud application is dependent on the completion time of coflows, rather than the completion time of each individual flow. Existing coflow-aware optimization solutions employ flow preemption to reduce the completion time, which brings difficulty in practical implementation and non-negligible overhead. In this paper, we study the non-preemptive coflow scheduling and routing problem in the cloud network. We propose an offline optimization framework for coflow scheduling, as well as two subroutines for coflow routing using single-path routing and multi-path routing respectively. We also show that our proposed framework is easily extensible to the online scenario. Extensive evaluations show that the proposed solutions can greatly reduce coflow completion time compared to coflow-agnostic solutions, and are also computationally efficient.**

Keywords*—*Coflow, scheduling and routing, flow completion time

I. Introduction

The cloud networks have become the performance bottleneck of many network-bound applications [1]. Most traditional network optimization solutions focus on optimizing application-agnostic network goals, such as end-to-end latency, flow completion time, or flow-level fairness. However, in practice, optimized application-agnostic network goals do not always translate into improved application performance. Many cloud applications still suffer from poor application-level communication performance [2].

Many data-intensive cloud applications typically have a *staged* computing process. Each stage of the application consists of multiple individual tasks. Stages are proceeded in sequence. The application proceeds to the next stage only when all tasks in the current stage are finished.

In each stage, a set of data flows may need to be transmitted before the final result can be generated. The transmission of those data flows can account for over 50% of the completion time of the application [3]. This urges for network optimization in terms of data flow groups rather than individual flows, for which the concept of *coflows* have been brought about [2]. A coflow defines a group of parallel individual flows. A coflow is finished only when all its component flows finish transmission. From the application's perspective, only the completion time of the entire coflow will affect the actual performance of the application. This is because even when some individual flows finish quickly, the application still needs to wait until the remaining flows finish transmission before it can proceed to the next stage.

Yu, Xue and Zhang ({ruozhouy, xue, xzhan229}@asu.edu) are all with Arizona State University, Tempe, AZ 85287. Tang (jtang02@syr.edu) is with Syracuse University, Syracuse, NY 13244. All correspondences should be addressed to Guoliang Xue. This research was supported in part by NSF grants 1457262. The information reported here does not reflect the position or the policy of the federal government.

To achieve coflow-level network optimization, two types of approaches can be employed. First, the *routing* of each component flow of a coflow needs to be determined in order to minimize its completion time. Second, as the network is commonly shared by a set of coflows, proper *scheduling* needs to be planned between coflows, in order to minimize the average completion time of all coflows.

In existing works [4]–[6], the scheduling of coflows is assumed to be *preemptive*. In other words, each flow (coflow) that is in transmission may be preempted by some other flow (coflow), if such preemption can result in reduced coflow completion time. Although preemption leads to better performance in theory, however, it is difficult to implement flow preemption without introducing any performance loss in practice. Consequences of flow preemption include non-negligible signaling overhead, extra latency for flow switching, dropped packets during flow preemption, etc.

Towards this end, in this paper we study the *non-preemptive* coflow scheduling and routing problem in the cloud network. We argue that, *once a flow starts transmission, neither its transmission path(s) nor its bandwidth allocation should be changed*. In this circumstance, we propose solutions for coflow scheduling and routing, in order to minimize the average coflow completion time in the network. Specifically, we first propose an offline optimization framework for the coflow scheduling and routing problem, which employs Shortest-Coflow First (SCF) as its basic scheduler. Based on the framework, we further propose two solutions for evaluating the coflow completion time of each coflow in the network: one using single-path routing, and one using multi-path routing. We also show that the offline optimization framework can be easily extended to an online scheduler, as shown in Section IV-D.

Our major contributions are summarized as follows:

- To the best of our knowledge, we are the first to study the coflow scheduling and routing problem in *non-preemptive* networks. The problem is formally defined.
- We propose an offline optimization framework for non-preemptive coflow scheduling and routing. Two subroutines are presented to find the minimum coflow completion time under single-path routing and multi-path routing respectively. The offline framework is easily extensible to the online scenario.
- We have conducted extensive simulations to evaluate the performance of the proposed solutions.

The rest of this paper is organized as follows. Section II introduces related work of coflow scheduling and routing in the literature. Section III presents the network model, and a formal description of the problem studied. Section IV presents the proposed solutions, including the offline optimization framework, the subroutines for computing minimum coflow completion time using single-path and multi-path routing, and the online algorithm extension. Section V shows the evaluation results. Section VI concludes this paper.

II. Related Work

Although the concept of coflow is new, there have been many researches regarding both flow scheduling and routing in cloud networks. In this section we introduce related work in terms of coflow and traditional flow scheduling and routing.

A. Coflow

Coflow is a newly developed concept, which introduces application-awareness in cloud network optimization. The coflow concept is first proposed in [2], where the authors analyzed the communication patterns of different cloud applications and argued to use grouped data flows as the network model of data parallel jobs. After that, two solutions Varys [4] and Aalo [6] both employ the coflow concept to develop flow scheduling solutions to minimize coflow completion time. Specifically, Varys [4] optimizes completion time given the size of each coflow (flow), while Aalo [6] assumes that the flow sizes are not known *a priori*. Rapier [5] follows the above intuitive for scheduling, but combines scheduling and coflow routing to further optimize coflow completion time. All the three works above employ preemptive flow scheduling to reduce coflow completion time. Barrat [7] also brings task-awareness into network optimization. Unlike the above works which are centralized, it proposes a distributed algorithm for task-aware flow scheduling.

Our work is most similar to Rapier [5], but differs in that we study the non-preemptive scheduling and routing problem.

B. Flow Scheduling

Flow and packet scheduling has been studied extensively in the context of cloud and data center networks to minimize flow completion time or meet flow deadlines [8]–[16]. DCTCP [8], D2TCP [10] and L2DCT [15] focus on improving flow completion time by modifying the default behavior of TCP at end-hosts, and thus do not require modification of the switch or host hardware. They rely on Explicit Congestion Notification (ECN) for congestion notification. On the other hand, D3 [9], PDQ [11], DeTail [12], HULL [13], pFabric [14] are clean-slate designs that require modification to switch hardware or host NICs. They rely on different flow and packet scheduling policies on the switch side to reduce congestion and improve completion time. PIAS [16] also does not require switch modification, but needs switches that have multiple priority queues. It employs end-host packet tagging for packet prioritization, and also relies on ECN for congestion notification.

C. Flow Routing

Flow routing or traffic engineering in cloud networks mainly focuses on load balancing, for example, [1], [17], [18]. Hedera [1] uses a centralized controller and several global routing algorithms for flow traffic engineering. Duet [18] further utilizes dedicated hardware to improve latency and availability upon software load balancers. Compared to them, the solutions proposed in this paper use explicit routing to achieve reduced coflow completion time.

III. Problem Statement

A. Network Model

We consider the cloud network as a directed graph $G = (V, E)$. V denotes the set of nodes. E denotes the set of links in the network, with c_e denoting the capacity of any link $e \in E$.

The network is concurrently shared by a set of data parallel jobs. Each job consists of multiple concurrent tasks, for which communication between network nodes is needed. The communication need of a job is modeled as a *coflow request*, which consists of multiple individual flow requests between different sources and destinations. The coflow request is completed only when all of its flow requests finish transmission.

Formally, the set of concurrent coflow requests is defined as $S = \{C_1, C_2, \ldots, C_m\}$. Each coflow request is defined as a set of flow requests, $C_i = \{F_{i,1}, F_{i,2}, \ldots, F_{i,n_i}\}$. Each individual flow request is defined as a triple, $F_{i,j} = (s_{i,j}, t_{i,j}, d_{i,j})$, where $s_{i,j} \in V$ is the source node, $t_{i,j} \in V$ is the destination node, and $d_{i,j} \in \mathbb{R}^+$ is the flow size.

B. Problem Statement

We study the problem of scheduling and routing of coflow requests. The primary objective is to minimize the average *coflow completion time* (CCT). Denote T_i as the CCT of coflow request C_i, and $T_{i,j}$ as the completion time of each individual flow request $F_{i,j}$. The CCT of coflow request C_i is defined as the maximum completion time of any of its flow requests:

$$T_i = \max_j \{T_{i,j} \,|\, F_{i,j} \in C_i\} \tag{1}$$

The completion time of each individual flow request has two components: transmission time and end-to-end delay. Since cloud networks commonly have ultra-low end-to-end delay [8], we neglect the end-to-end delay in the computation of flow completion time. The transmission time of a flow request is determined by the aggregated bandwidth that it receives over time. Formally, define $B_{i,j}(t) : \mathbb{R}^* \mapsto \mathbb{R}^*$ as the instantaneous bandwidth received by $F_{i,j}$ at time t, its completion time is defined as

$$T_{i,j} = \arg\min_{\tau} \left\{ \int_0^{\tau} B_{i,j}(t)\, dt = d_{i,j} \right\} \tag{2}$$

We denote $B_{i,j}(t)$ as the *bandwidth function* of flow $F_{i,j}$.

The bandwidth function of a flow implicitly defines the scheduling of the flow. Two types of flow scheduling techniques exist in the literature: *preemptive scheduling* and *non-preemptive scheduling*. Compared to non-preemptive scheduling, preemptive flow scheduling is a recently proposed technique that allows a newly arrived flow to preempt the transmission of an already scheduled flow, thus to reduce flow completion time by preferring shorter flows [11]. However, while preemptive scheduling can result in shorter flow completion time in theory, it encounters many challenges in implementation, including extra signaling overhead for flow preemption, introduced flow switching latency, packet drops during the preemption process, requirement for designated switch hardware, etc. As a result, flow preemption is far from being practically applied to production cloud networks. Hence in this paper, we assume that the scheduling of flows is *non-preemptive*, meaning that the data path and bandwidth share of a flow will remain unchanged once it starts transmission.

Definition 3.1: A bandwidth function $B_{i,j}(t)$ is said to be *non-preemptive* if it is in the following form:

$$B_{i,j}(t) = \begin{cases} b_{i,j}\,, & t \in [t_s, t_e) \\ 0\,, & t < t_s \text{ or } t \geq t_e \end{cases} \tag{3}$$

where t_s and t_e are the start and end time of $F_{i,j}$ respectively, and $b_{i,j}$ is the constant bandwidth received by $F_{i,j}$. □

In other words, the transmission of $F_{i,j}$ will receive constant bandwidth $b_{i,j}$ from start time t_s, until its completion at t_e.

In the network, each flow request is realized by actual network paths. We consider both single-path routing and multi-path routing for flow requests. The set of network paths used by flow request $F_{i,j}$ is denoted by $P_{i,j}$. Since the network is assumed to be non-preemptive, the path(s) of each flow request cannot be changed once it starts transmission.

The bandwidth of $F_{i,j}$ is defined as the sum of bandwidth from all paths in $P_{i,j}$. Formally, for each path $p \in P_{i,j}$, we use $B^p_{i,j}(t)$ to denote the bandwidth function of flow $F_{i,j}$ on path p, and we have

$$B_{i,j}(t) = \sum_{p \in P_{i,j}} B^p_{i,j}(t) \tag{4}$$

We use $\mathbf{B}_{i,j}(t)$ to denote the set of bandwidth functions on all paths for $F_{i,j}$, given $P_{i,j}$.

Similar to Eq. (3), we say that bandwidth function $B^p_{i,j}(t)$ is *non-preemptive* iff flow $F_{i,j}$ receives constant bandwidth $b^p_{i,j}$ on path p from t_s to t_e, and 0 bandwidth at all other times.

Given above, the *Non-preemptive Coflow Scheduling and Routing* (NCSR) problem is defined as follows:

Definition 3.2 (NCSR): Given network $G = (V, E)$ and coflow requests S, find a path set $P_{i,j}$ and non-preemptive bandwidth functions $\mathbf{B}_{i,j}(t)$ for each flow $F_{i,j}$, such that 1) at any time t, the total bandwidth allocated on any link e does not exceed its capacity c_e, and 2) the average CCT of all coflow requests is minimized. □

Note that since the number of CCTs is fixed, minimizing the average CCT is equivalent to minimizing the sum of CCT of all coflows, *i.e.*,

$$\min \quad \sum_{C_i \in S} T_i\,. \tag{5}$$

IV. Non-preemptive Scheduling and Routing

A. *Offline Optimization Framework*

We first state our offline optimization framework for non-preemptive coflow scheduling and routing. The essence of our framework is a Shortest-Coflow First (SCF) scheduler, which schedules coflow requests according to their minimum CCT. The framework is shown in Algorithm 1.

Algorithm 1: Coflow optimization framework

```
  Input: topology $G$ and coflow requests $S$
  Output: per-flow paths $P_{i,j}$ and bandwidth functions
          $\mathbf{B}_{i,j}(t)$ for $\forall F_{i,j} \in C_i$
1 for each coflow request $C_i \in S$ do
2     if network supports multi-path routing then
3         $[\mathcal{T}_i, P_{i,j}, b_{i,j}] \leftarrow multiCCT(G, C_i)$;
4     else
5         $[\mathcal{T}_i, P_{i,j}, b_{i,j}] \leftarrow singleCCT(G, C_i)$;
6     end
7 end
8 Schedule all coflows in ascending order of $\mathcal{T}_i$;
9 return $P_{i,j}$ and $\mathbf{B}_{i,j}(t)$ based on $b_{i,j}$ and scheduling.
```

To compute the minimum CCT for each coflow request, the algorithm relies on two subroutines: $singleCCT$ when only single-path routing is enabled, and $multiCCT$ when multi-pathing is allowed. As the network is non-preemptive, no two coflow requests will share the network at the same time, in order to avoid bandwidth competition among coflow requests, which may result in prolonged coflow completion time for each coflow. After computing the minimum CCT, all coflow requests are scheduled using Shortest-Coflow First.

B. *Multi-path Routing*

In modern cloud networks, multi-pathing is a common practice to reduce congestion and increase bandwidth utilization [19]. In this subsection, we present $multiCCT$, which finds the minimum CCT for each coflow request using multi-path routing. The input of the subroutine is the network topology G, and a coflow request C_i. This is because during the transmission of C_i, it exclusively uses all the bandwidth in the network.

A network flow-based formulation of the problem is formulated as follows. Define the following variables:

1) $f^e_{i,j} \in \mathbb{R}^+$: the flow value of $F_{i,j}$ on edge e.
2) $b_{i,j} \in \mathbb{R}^+$: the constant bandwidth value of $F_{i,j}$.
3) $\mathcal{T}_i \in \mathbb{R}^+$: the (tentative) CCT of coflow request C_i.

The problem of finding the minimum CCT of coflow request C_i can be formulated as follows:

$$\min \quad \mathcal{T}_i \tag{6a}$$

$$\text{s.t.} \quad \mathcal{T}_i = \max_j \{d_{i,j}/b_{i,j}\} \tag{6b}$$

$$\sum_{j=1}^{n_i} f^e_{i,j} \leq c_e \qquad \forall e \in E \tag{6c}$$

$$\sum_{(u,v) \in E} f^{(u,v)}_{i,j} - \sum_{(v,w) \in E} f^{(v,w)}_{i,j} = \begin{cases} 0\,, & v \notin \{s_{i,j}, t_{i,j}\} \\ -b_{i,j}\,, & v = s_{i,j} \\ b_{i,j}\,, & v = t_{i,j} \end{cases}$$
$$\forall F_{i,j} \in C_i, v \in V \tag{6d}$$

Objective (6a) minimizes the tentative CCT $\mathcal{T}_i$ of coflow request C_i. Note that $\mathcal{T}_i \neq T_i$, as the final completion time of C_i is also determined by the scheduling among coflow requests. Constraint (6b) defines the CCT of the coflow request, as the maximum ratio of $d_{i,j}/b_{i,j}$ among all component flows, where $b_{i,j}$ is the aggregated constant bandwidth of $F_{i,j}$.

Constraint (6c) specifies the capacity constraint of each edge. Constraint (6d) specifies flow conservation constraints at each node, in which the aggregated bandwidth of each component flow is defined.

This formulation is non-linear due to the existence of Constraint (6b). However, it can be transformed into an equivalent linear program as follows:

$$\max \quad f_i \tag{7a}$$
$$\text{s.t.} \quad f_i \leq b_{i,j}/d_{i,j} \qquad \forall F_{i,j} \in C_i \tag{7b}$$
$$\text{(6c) and (6d)}$$

where $f_i = 1/\mathcal{T}_i$.

Since (7) is a linear program, it can be solved in polynomial time. After that, the resulting flows can be easily decomposed into sets of network paths, with proper bandwidth allocations.

C. Single-path Routing

Multi-path routing can increase the available bisectional bandwidth of each flow, and thus potentially reduce the coflow completion time. However, not all network environments support multi-path routing by default. In many cases, each flow request can only be realized by one network path. In this subsection, we introduce subroutine $singleCCT$, which computes the minimum CCT using single-path routing.

Formally, define the following variables:

1) $x^e_{i,j} \in \{0, 1\}$: the indicator of whether edge e is included in the path of $F_{i,j}$ ($x^e_{i,j} = 1$) or not ($x^e_{i,j} = 0$).
2) $b_{i,j} \in \mathbb{R}^+$: the constant bandwidth value of $F_{i,j}$.
3) $\mathcal{T}_i \in \mathbb{R}^+$: the (tentative) CCT of coflow C_i.

The problem of coflow scheduling and single-path routing can be formulated as the following mixed integer program:

$$\min \quad \mathcal{T}_i \tag{8a}$$
$$\text{s.t.} \quad \mathcal{T}_i = \max_j \{d_{i,j}/b_{i,j}\} \tag{8b}$$
$$\sum_{j=1}^{n_i} x^e_{i,j} \cdot b_{i,j} \leq c_e \qquad \forall e \in E \tag{8c}$$
$$\sum_{(u,v)\in E} x^{(u,v)}_{i,j} - \sum_{(v,w)\in E} x^{(v,w)}_{i,j} = \begin{cases} 0, & v \notin \{s_{i,j}, t_{i,j}\} \\ -1, & v = s_{i,j} \\ 1, & v = t_{i,j} \end{cases}$$
$$\forall F_{i,j} \in C_i, v \in V \tag{8d}$$

Objective (8a) is to minimize the tentative coflow completion time $\mathcal{T}_i$. Constraint (8b) defines the completion time $\mathcal{T}_i$ as the flow size $d_{i,j}$ divided by bandwidth $b_{i,j}$. Constraint (8c) specifies the capacity constraints on each edge. Constraint (8d) is the single-path conservation constraint.

The above formulation is a non-linear mixed integer program, due to the non-linearity of Constraints (8b) and (8c), and the integral constraints on variables $x^e_{i,j}$. Hence it is intractable to directly solve the program. To reduce the complexity for solving the program, we apply the following transformations.

First, we eliminate the non-linearity of Constraint (8b) using a similar method as in the previous subsection. The resulting program is as follows:

$$\max \quad f_i \tag{9a}$$
$$\text{s.t.} \quad f_i \leq b_{i,j}/d_{i,j} \qquad \forall F_{i,j} \in C_i \tag{9b}$$
$$\text{(8c) and (8d)}$$

Next, we linearize program (9) by introducing a new flow variable $f^e_{i,j} = x^e_{i,j} \cdot b_{i,j}$. Unlike $x^e_{i,j}$, variable $f^e_{i,j}$ can take any value in $\mathbb{R}^+$. By replacing all occurrences of $x^e_{i,j}$ with $f^e_{i,j}/b_{i,j}$, and multiplying Constraint (8d) by $b_{i,j}$ on both sides, we obtain a linear program. Since we relax the integral constraints on variables, the resulting program is a *linear relaxation* of the original program (9a). One may observe that the resulting program is actually identical to the program in (6). Exploiting the linear relaxation, the $singleCCT$ subroutine is shown in Algorithm 2.

Algorithm 2: $singleCCT(G, C_i)$

```
Input: topology $G$ and coflow request $C_i$
Output: minimum CCT $\mathcal{T}_i$, per-flow paths $P_{i,j}$ and
        bandwidth allocation $b_{i,j}$ for $\forall F_{i,j} \in C_i$
1 Solve the linear relaxation of program (9);
2 for each flow request $F_{i,j} \in C_i$ do
3     Find path $p_{i,j} \leftarrow \arg\max_p \{\xi^p_{i,j}\}$ where
      $\xi^p_{i,j} = \min\{x^e_{i,j} | e \in p\}$, from $s_{i,j}$ to $t_{i,j}$;
4     $P_{i,j} \leftarrow \{p_{i,j}\}$;
5     Set $x^e_{i,j}$ to 1 for $\forall e \in p_{i,j}$, and 0 for $\forall e \notin p_{i,j}$;
6 end
7 Given fixed values of $x^e_{i,j}$, solve (9) to obtain $b_{i,j}$ for
  each $F_{i,j}$, and $f_i$;
8 return $\mathcal{T}_i = 1/f_i$, $P_{i,j}$ and $b_{i,j}$.
```

The algorithm first solves the linear relaxation to obtain a fractional solution. After that, it employs deterministic rounding to find an integral solution to per-flow path selection. Specifically, for each flow request, it finds the widest $(s_{i,j}, t_{i,j})$-path $p_{i,j}$ with regard to weights denoted by $x^e_{i,j}$. This path is then assigned to the flow request. After all flow requests are routed, the bandwidth allocation is then computed by solving (9) again, using the fixed path assignments. Note that with fixed variables $x^e_{i,j}$, program (9a) now becomes a linear program, and thus can be solved in polynomial time.

D. Online Scheduling and Routing

The framework proposed in Section IV-A is an offline optimization framework. In this subsection, we modify the framework to enable online coflow optimization.

At any time, the network scheduler maintains a list of coflow requests that have yet been scheduled, along with the minimum CCT of each coflow request. Upon the completion of the current coflow request, the next coflow request with the minimum CCT will be scheduled. When a new coflow request arrives, the scheduler will first compute its minimum CCT using either $singleCCT$ or $multiCCT$. It then inserts the new coflow request into the list of non-scheduled coflows.

To avoid starvation of large-size coflow requests, a waiting threshold Γ can be defined. Any coflow request that has waited for more than Γ time will be scheduled immediately in the next slot, after the completion of the current request.

V. Performance Evaluation

A. Experiment Methods

We use randomly generated topologies and coflow requests to evaluate the performance of the proposed solutions. The random topologies are generated according to a modified Waxman model, with connectivity guarantee. The default parameters used in the Waxman model are $\alpha = 0.15$ and $\beta = 0.2$. Each topology by default contains 50 network nodes. Link capacities are uniformly generated from $[10, 100]$ Mbps. For each experiment, we randomly generate 25 coflow requests. Each coflow randomly has 1 to 10 component flows by default. Flow sizes are uniformly generated from $[10, 100]$ Mb.

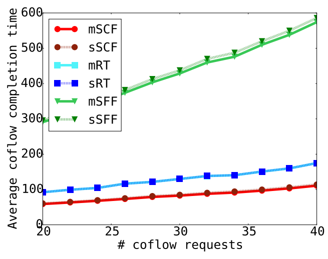
In this paper, we have proposed solutions for non-preemptive coflow scheduling and routing, which we name as `SCF` (`sSCF` for single-path version and `mSCF` for multi-path version). We compare our algorithms to a routing-only algorithm, as well as a non-preemptive baseline solution for coflow-agnostic flow scheduling and routing, listed as follows:

- **Routing-only** (`RT`): all flows are routed concurrently, with the objective of minimizing maximum CCT.
- **Shortest-Flow First** (`SFF`): each flow is routed exclusively in the network, and flows are scheduled with Shortest-Flow First.

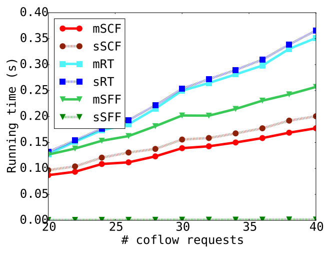
Both algorithms have two versions: for single-path routing (`sRT` and `sSFF`) and for multi-path routing (`mRT` and `mSFF`) respectively. `mRT`, `sRT` and `mSFF` use similar linear programs for multi-path routing, and `sRT` uses deterministic rounding for single-path routing. `sSFF` uses modified Dijkstra's algorithm to compute the widest path for each flow, and uses Shortest-Flow First for scheduling. Detailed descriptions are omitted in this paper due to the page limit.

All experiments are conducted on a Macbook Air, with Intel Core i7 CPU (1.7GHz) and 8GB main memory. The Gurobi [20] solver is used to solve linear programs in the algorithms. Each experiment is run for 50 times under the same experiment setting, and the following results are the average of all the runs under the same settings.

B. Evaluation Results



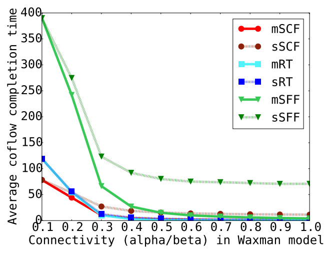
(a) Average CCT vs. # coflow requests



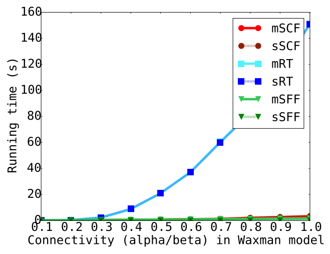
(b) Running time vs. # coflow requests

Fig. 1: Average coflow completion time and running time versus number of coflow requests.

We have conducted experiments with various settings to evaluate the performance of our proposed algorithms. In Fig. 1, we show the results under various number of coflow requests in the network. Fig. 1(a) shows the average CCT versus the number of coflow requests. Both `mSCF` and `sSCF` outperform the other algorithms. The `SFF` algorithms have very high average CCT due to their coflow-unawareness. The `RT` algorithms have higher average CCT than `SCF` algorithms because they do not schedule between short and long coflows. All algorithms have average CCT increasing with the number of coflow requests, i.e., as the network becoming more and more congested. Fig. 1(b) shows the running time of the algorithms. `SCF` algorithms have lower running time than `RT` in that they solve many smaller linear programs each time, while `RT` algorithms solve an aggregated linear program with more variables and constraints. `sSFF` has the lowest running time as it does not involve solving linear programs but rather uses the Dijkstra's algorithm to compute widest paths.



(a) Average CCT vs. network connectivity



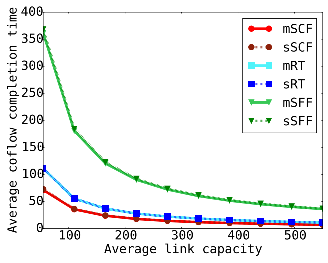
(b) Running time vs. network connectivity

Fig. 2: Average coflow completion time and running time versus network connectivity.

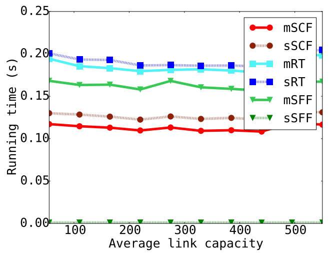
Fig. 2 shows the results against various levels of connectivity of the network topology, determined by the two arguments α and β used in the Waxman model for topology generation. Higher values of both α and β means higher connectivity in the network. Fig. 2(a) shows the average CCT against network connectivity. Average CCT decreases with increased network connectivity, due to the increase in bisectional bandwidth between any pair. With higher connectivity, multi-path routing performs better than single-path routing, as it can utilize more bandwidth in the network. Also, as the network becomes more and more spare, `RT` algorithms can utilize more bandwidth by bandwidth sharing among coflows, and thus outperform `SCF` algorithms. However, as bandwidth is a scarce resource in cloud networks, such scenario is not common. `SFF` algorithms perform the worst due to their coflow-unawareness. As for the running time shown in Fig. 2(b), all algorithms have increasing running time with increasing connectivity. The running time of `RT` algorithms dominates the others as they solve an aggregated linear program, which is time consuming.

Fig. 3 shows the results against various average link capacities. Fig. 3(a) shows the average CCT against link capacities. Average CCT decreases with increased link capacities. `SCF` algorithms still outperform `RT` and `SFF` algorithms. `SFF` algorithms perform the worst due to their coflow-unawareness. In Fig. 3(b), since both the network size and the number of coflows remain unchanged, the running time remains at the same level with increasing link capacities.

Fig. 4 shows the results against the number of nodes in the network. Fig. 4(a) shows the average CCT against the number of nodes. As shown, the average CCT first remains steady, but then decreases with increasing number of nodes.

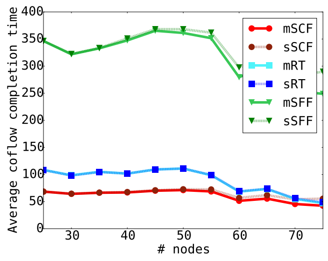


(a) Average CCT vs. average link cap.

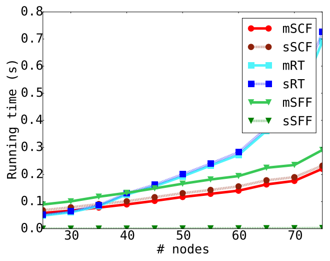


(b) Running time vs. average link cap.

Fig. 3: Average coflow completion time and running time versus average link capacity.



(a) Average CCT vs. # nodes



(b) Running time vs. # nodes

Fig. 4: Average coflow completion time and running time versus # nodes in the network.

The steady phase is due to that the network is fully congested by the coflows. With further increased network size, the network becomes less congested in the second phase, and thus the average CCT reduces. One can observe that when the network is congested, `SCF` outperforms `RT` by scheduling shorter coflows first. Such advantage does not hold when the network is spare, as in this case `RT` can utilize more bandwidth through bandwidth sharing among coflows. As bandwidth is commonly a scarce resource in the network, `SCF` algorithms are expected to outperform `RT` algorithms in most cases. As baseline, `SFF` algorithms still perform the worst due to their coflow-unawareness. In Fig. 4(b), the running time of all algorithms increases with the size of the network.

In summary, we make the following conclusions.

1) `SCF` algorithms outperform `RT` algorithms when the network is congested, while the contrary happens when the network is spare. As the cloud networks are commonly congested, we anticipate that `SCF` algorithms will outperform `RT` algorithms in practical environments. On the other hand, the `RT` algorithms can serve as a backup plan for `SCF` when the network is not congested.
2) `SCF` has lower time complexity than `RT` due to the smaller-sized linear programs that it solves. This makes `SCF` more suitable as an online algorithm compared to `RT`.
3) With coflow-awareness, both `SCF` and `RT` greatly outperform `SFF` regarding the average coflow completion time.

VI. Conclusions

In this paper, we studied the non-preemptive coflow scheduling and routing problem in the cloud network. We proposed an offline optimization framework for non-preemptive coflow scheduling and routing, which is based on a Shortest-Coflow First scheduler. To implement the framework, we proposed two subroutines, $singleCCT$ and $multiCCT$, to compute the minimum coflow completion times of each coflow, using single-path routing and multi-path routing respectively. Our simulation results show that the proposed algorithms greatly improve the average coflow completion time over routing-only solutions or coflow-agnostic non-preemptive scheduling solutions, and also have higher time efficiency.

References

[1] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera : Dynamic Flow Scheduling for Data Center Networks," in *USENIX NSDI*, 2010.

[2] M. Chowdhury and I. Stoica, "Coflow: a networking abstraction for cluster applications," in *ACM HotNets*, 2012.

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

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

[5] Y. Zhao, K. Chen, W. Bai, M. Yu, 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.

[6] M. Chowdhury and I. Stoica, "Efficient Coflow Scheduling Without Prior Knowledge," in *ACM SIGCOMM*, 2015.

[7] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized Task-Aware Scheduling for Data Center Networks," in *ACM SIGCOMM*, 2014.

[8] M. Alizadeh, A. Greenberg, D. a. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *ACM SIGCOMM*, 2010.

[9] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never than Late : Meeting Deadlines in Datacenter Networks," in *ACM SIGCOMM*, 2011.

[10] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (D2TCP)," in *ACM SIGCOMM*, 2012.

[11] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *ACM SIGCOMM*, 2012.

[12] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: reducing the flow completion time tail in datacenter networks," in *ACM SIGCOMM*, 2012.

[13] M. Alizadeh, A. Kabbani, T. Edsall, and B. Prabhakar, "Less is More : Trading a little Bandwidth for Ultra-Low Latency in the Data Center," in *USENIX NSDI*, 2012.

[14] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: minimal near-optimal datacenter transport," in *ACM SIGCOMM*, 2013.

[15] A. Munir, I. a. Qazi, Z. a. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *IEEE INFOCOM*, 2013.

[16] 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.

[17] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *ACM CoNEXT*, 2011.

[18] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud Scale Load Balancing with Hardware and Software," in *ACM SIGCOMM*, 2014.

[19] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley, "Data center networking with multipath TCP," in *ACM HotNets*, 2010.

[20] G. Optimization, "Gurobi Optimizer." [Online]. Available: http://www.gurobi.com/products/gurobi-optimizer