# Far-sighted Multi-stage Aware Coflow Scheduling

Shuai Zhang†, Sheng Zhang†, Xiaoda Zhang†, Zhuzhong Qian†
Mingjun Xiao§, Jie Wu‡, Jidong Ge†, Xiaoliang Wang†
†State Key Lab. for Novel Software Technology, Nanjing University, P.R. China
§University of Science and Technology of China, P.R. China
‡Center for Networked Computing, Temple University
Email: 151220162@smail.nju.edu.cn, sheng@nju.edu.cn

***Abstract*—In data center networks (DCN), large scale flows produced by parallel computing frameworks form many coflows semantically. Most inter-coflow schedulers only focus on the remaining data of coflows and attempt to mimic Shortest Job First (SJF). However, a coflow may consist of multiple stages. In this paper, we consider the Multi-stage Inter-Coflow Scheduling problem and try to give an efficient online scheduling scheme. We first explore a short-sighted algorithm with the greedy strategy. This gives us an insight into utilizing the network resources. Based on that, we propose a far-sighted heuristic, which schedules sub-coflows to occupy network bandwidth in turn. Through simulations in various network environments, we show that, compared to a state-of-the-art scheduler – Varys, a multi-stage aware scheduler can reduce the coflow completion time by up to 4.81× even though it is short-sighted. Moreover, the far-sighted scheduler can improve the performance by nearly 7.95× reduction.***

***Index Terms*—coflow, scheduling, multi-stage**

## I. Introduction

A huge demand for the processing of big data has hastened the birth of cluster computing frameworks such as MapReduce [1], Spark [2], Dryad [3] and so on. During the period of processing, a large amount of data is produced and needs to be transferred in data center network. The transmission of data accounts for a significant part in task completion time, which becomes the focus of data center researches in the recent years. Previous researches [4–8] have studied the flow scheduling, in an attempt to decrease the flow completion time (FCT). However, a task of one application may produce many flows which are interdependent. Scheduling flows in isolation would then lose their communication semantics.

The recently proposed *coflow* abstraction enables semantic-aware scheduling [9]. A coflow refers to a set of flows that share the same performance target. Flows in a coflow serve the same task and have dependency on each other. Completion of a coflow requires the completion of all flows in it. In virtue of this abstraction, scheduler is able to be aware of the requirements of applications, obtaining a better coflow completion time (CCT). Unfortunately, minimizing CCT by inter-coflow scheduling is NP-hard [10]. As a consequence, some efficient heuristic algorithms have emerged [10–14].

Considering that Shortest Job First (SJF) is the optimal scheduling scheme for minimizing the average FCT over a single link, many heuristics [4, 5, 7, 10] try to emulate SJF. Most of them prioritize a coflow by counting the accumulative data it has sent. Despite the fact that this method effectively distinguishes coflows, we must notice that a coflow may consist of multiple stages, and different stages transfer different amounts of data [15]. Some coflows send a lot of data in earlier stages, and then their to-be-transmitted data would decrease. But before the decrease, traditional heuristics have deprioritized these coflows because their accumulative transmission data excesses the threshold and these coflows are identified as large ones. As a consequence, their bandwidth is narrow even if their remaining data is much less than other coflows. Obviously, this mismatch will certainly elongate the average CCT.

There exist many stages during the processing of a task which has a corresponding coflow. We designate a stage of the task associated with one coflow as a sub-coflow. It has two phases: transmission and computation. During the first one, all the flows belonging to this coflow transfer data from sources to destinations. After all these flows finish transmission, the computation begins. When the second phase finishes, this sub-coflow is complete accordingly. In parallel computing frameworks, the subsequent stages have dependency on the antecedent stages, so in a coflow, the beginning of each sub-coflow requires the completion of all the antecedent sub-coflows.

In this paper, we propose the *Multi-stage Inter-Coflow Scheduling* (MICS) Problem and present a multi-stage aware scheduling framework. With the multi-stage awareness, schedulers are able to have a more fine-grained view of coflow than traditional heuristics. Moreover, we introduce two heuristics to optimize the average CCT: *Iteratively Approaching Optimal* (IAO) and *Multi-stage Least Bottleneck First* (MLBF). The former is beneficial to guarantee fairness among coflows while the latter can deliver a better performance.

In IAO, we partition the original MICS problem into many *Single-stage Inter-Coflow Scheduling* (SICS) Problem. In each SICS problem, we greedily minimize the average CCT of corresponding sub-coflows, and then combine the scheduling result in each SICS problem into the final result for MICS. We first prove that the SICS problem is a convex optimization problem, which allows us to search for the optimal solution. In virtue of interior point method [16], we design IAO to minimize the average sub-coflow completion time (SCCT). In our experiments, IAO outperforms significantly a non multi-stage aware one (e.g., Varys [10]). As the size of coflows increases exponentially, it improves both average CCT (from $2\times$ to $7.95\times$) and the completion time of all coflows (from $2.6\times$ to $7.42\times$) compared to Varys.

However, IAO is short-sighted to some extent. It only focuses the current sub-coflows, leading to a local optimal solution but not optimal for the entire transmission of all the coflows. The lack of foresight affects the performance of IAO, since IAO tends to transfer the current sub-coflows simultaneously and consequently gives rise to overlap of their computation phases. During the overlapping period, network is idle, which is a great waste of link bandwidth.

In order to improve the network utilization, network should be used by one sub-coflow exclusively instead of sharing. Based on SJF, we propose MLBF heuristic to achieve this target. In our experiment, it outperforms IAO from $1\times$ to $1.65\times$ in average CCT with the exponential increase in the number of coflows.

The rest of this paper is organized as follows. We introduce the network model, the coflow abstraction and the definition of sub-coflow in Section II. Furthermore, we formulate the MICS problem and prove its harness in Section III. Then we propose two heuristics to solve this problem in Section IV and evaluate them in Section V. Last but not least, we discuss our work in Section VI, present related work in Section VII and conclude this paper in Section VIII.

## II. Background

### A. Network Model

In our study, we consider the data center network as a giant non-blocking switch [4, 10, 13, 17] and only focus on its ingress and egress ports (Figure 1). This abstraction is reasonable because of the great progress made in full bisection bandwidth topologies [18]. In this model, an ingress port might send data to one or more egress ports, and an egress port might receive data from one or more ingress ports. Owing to the limited bandwidth of switch ports, we need to allocate bandwidth to flows between ingress port set $\mathbb{P}$ and egress port set $\mathbb{Q}$. In Figure 1, there are more than one coflow passing through an ingress port and there is only one flow of each coflow in a port.

### B. Coflow Abstraction

A coflow refers to a set of flows that are produced by two groups of machines which serve the same task and share the same performance target in the cluster parallel computing framework. We define a flow $f$ of coflow $C_i$ as a tuple $f = (p, q) \in C_i$, where $p \in P_i$ and $q \in Q_i$ are the source and destination ports respectively. In our network model, coflow $C_i$ has ingress port set $P_i \subseteq \mathbb{P}$ and egress port set $Q_i \subseteq \mathbb{Q}$. A coflow has multiple stages, and the beginning of each stage requires the completion of all the antecedent stages. A stage of one coflow is defined as a sub-coflow. The set of coflows in the network is $\mathbb{C} = \{C_1, C_2, \ldots, C_n\}$, and every coflow has $\phi$ stages, which means each coflow has $\phi$ interdependent sub-coflows.

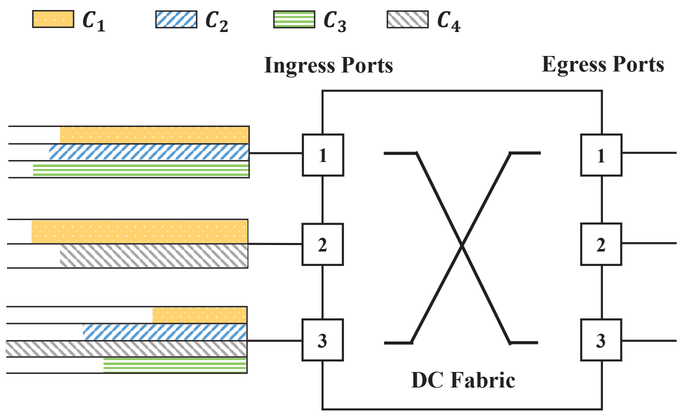

**Figure 1.** Coflow scheduling over Date Center Fabric with $3\times3$ ingress/egress ports. There exist 4 coflows: $C_1$, $C_2$, $C_3$, and $C_4$. The width of flows indicates the rate of flow and the width of ingress ports shows the bandwidth limit of ports. Areas of flows are their remaining data.

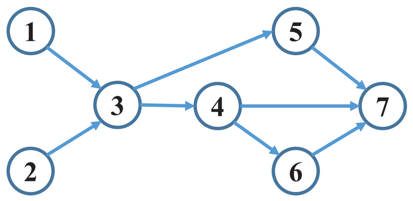

**Figure 2.** The dependency between sub-coflows in coflow $C_i$. No.$j$ is sub-coflow $C_{i,j}$ and the arrows show their processing sequence.

### C. Sub-Coflow

A coflow is composed of multiple sub-coflows in a certain sequence, and flows in these sub-coflows are the same. The $j$-th stage of $C_i$ is sub-coflow $C_{i,j}$. Once all the sub-coflows on which $C_{i,j}$ depends complete, $C_{i,j}$ would begin immediately. For example, in Figure 2, sub-coflow $C_{i,3}$ would not begin until both $C_{i,1}$ and $C_{i,2}$ complete. Sub-coflows are processed in topological order in this DAG.

$C_{i,j}$ begins transferring at $t_{i,j}$, and after the completion of all its flows, it waits for the computing phase. Computation would generate new data, which is the transmission load of the next sub-coflow. When the computing phase finishes, the sub-coflow completes and we define sub-coflow's completion time as SCCT. During the period of computation, the ports of $C_{i,j}$ are idle. We can assume that all the coflows arrive at the same time 0, namely, $t_{i,1} = 0$ $(\forall 1 \leq i \leq n)$.

## III. Problem Formulation

Given the information about a set of coflows and the bandwidth of switch ports, we are supposed to decide *at what rate* to serve their flows at a certain time to minimize the average coflow completion time (CCT). In other words, if we already know the data to be transferred for $\forall f = (p, q) \in C_{i,j}$ is $V_{i,j,p,q}$, the related computing time is $T_{i,j}$ and the bandwidth of $p$ and $q$ are $R_p$ and $R_q$, a function of time $r_{i,j,p,q}(t)$ is required that describes the rate allocation for flow $f = (p, q) \in C_{i,j}$ at time $t$.

In DCN, coflow scheduling is subject to the following restrictions:

1. *Bandwidth Constraints*. At any time $t$, the total bandwidth allocated to any port should not be more than its capacity:

$$\sum_{i,j,q} r_{i,j,p,q}(t) \leq R_p, \ \forall t \tag{1}$$

$$\sum_{i,j,p} r_{i,j,p,q}(t) \leq R_q, \ \forall t \tag{2}$$

2. *Order of sub-coflows*. For every coflow, its sub-coflows must be processed in topological order in dependency graph. $C_{i,j}$ can begin to be processed only after all of $C_{i,h}$ it depends on have completed. Then the constraint of dependency is as follows:

$$t_{i,h} + \max_{p,q} \tau_{i,h,p,q} + T_{i,h} \leq t_{i,j}, \ \forall h \in \{x | C_j \text{ depends on } C_x\} \tag{3}$$

In this inequation, $\tau_{i,h,p,q}$ is the time cost of transferring the data of flow $f = (p, q) \in C_{i,h}$ and can be calculated by:

$$\int_0^{\tau_{i,h,p,q}} r_{i,h,p,q}(t)dt = V_{i,h,p,q} \tag{4}$$

Considering that we only focus on the transmission of data, we define CCT$(C_i)$ as the completion time of the data transferring phase of the last sub-coflow $C_{i,\phi}$.

$$\text{CCT}(C_i) = t_{i,\phi} + \max_{(p,q)} \tau_{i,\phi,p,q}$$

As a result, the target is:

$$\min \ \frac{1}{n} \sum_{i=1}^{n} \text{CCT}(C_i)$$

We model this scheduling problem with *Multi-stage Inter-Coflow Scheduling Problem* (MICS) and provide an online scheduling framework.

**Theorem** 1: The MICS problem is NP-hard.

*Proof:* Chowdhury *et al.* [10] have proved that in the same network model, scheduling of single-stage coflows to minimize average CCT is NP-hard. MICS problem is the scheduling of a combination of many single-stage coflows. As a result, minimizing average CCT of multi-stage coflows is NP-hard. □

## IV. Online Scheduling

Before the scheduling scheme, we firstly propose an online scheduling framework: At the beginning, initial rate of $C_{i,1}$ ($\forall 1 \leq i \leq n$) would be allocated at time 0. Rate allocation only happen when a new sub-coflow arrives or an old one completes. In experiments, we find it fairly balances the network utilization and the overhead of frequent scheduling. In addition, several assumptions are introduced to make us focus on important decisions.

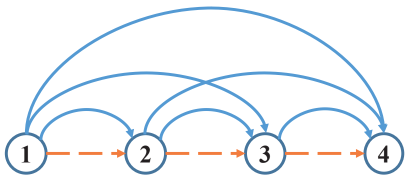

**Figure 3.** The solid (blue) arrows indicate the processing sequence that sub-coflows of the same coflow should follow. The dashed (orange) arrows show the actual sequence.

### A. Assumptions

1. *Identical Switch Port*. In consideration of the large-scale deployment of routers in data center, most of their transmission capacities are similar. In our abstraction, we can assume that all the capacities of switch ports are the same bandwidth $R$.

2. *Bijection*. We define a flow $f = (p, q)$ a bijection from ingress port $p$ to egress port $q$, which means that, in a coflow, each ingress port only receives data from one egress port, and vice versa. Given that each flow has a unique ingress port and egress port in a coflow, different flows would not conflict each other on any port. By this way, we no longer need to worry about the rate division of flows that share the same port. Based on this assumption, we directly use $f$ to represent a flow, instead of 2-tuple $(p, q)$.

3. *Rate Consistency*. Between two schedules, the rate of one flow should not change, which is guaranteed by the framework schedule strategy.

4. *Ascending Sequence*. In one coflow $C_i$, sub-coflow $C_{i,j}$ depends on all of $C_{i,h}(1 \leq h < j)$ and none of $C_{i,h}(h \geq j)$. Under this assumption, there is no ring in the dependency graph and an inference can be drawn that the next sub-coflow $C_{i,j}$ can begin immediately once the previous $C_{i,j-1}$ completes. For example, as shown in Figure 3, each sub-coflow depends on all its antecedent ones, so the only feasible sequence is 1, 2, 3, 4. This assumption ensures that the sequence of sub-coflows in $C_i$ is $C_{i,1}, C_{i,2}, ..., C_{i,\phi}$.

### B. Problem Analysis

With this framework, we only focus on a simplified problem: Allocate rate $r_{i,f}$ for all the flows in the current sub-coflow of $C_i$ before the next scheduling. Based on a simple greedy, we try to minimize the average SCCT of current sub-coflows, which will finally lead to the decrease of average CCT. We assume that, the current sub-coflow of $C_i$ is $C_{i,s_i}$, the completion time of $C_{i,s_i}$ is $t_i$, and the remaining volume of its flow $f$ is $V_{i,f}$. Then Equation (4) turns into:

$$t_i = \max_f \frac{V_{i,f}}{r_{i,f}} \tag{5}$$

Our online scheduling target is minimizing function $T(r)$, the sum of completion time of all the current sub-coflows.

$$T(r) = \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} \max_f \frac{V_{i,f}}{r_{i,f}} \tag{6}$$

The aggregate rate of all the current flows should not exceed the capacities of ports. Note that based on the aforementioned definition, flow is equivalent to a source-destination port pair. For convenience, we denote $\omega$ as the number of flows in $\bigcup_{i=1}^{n} C_{i,s_i}$, which is the set of flows of current sub-coflows. The original constraints Equation (1) and (2) can be simplified to:

$$\sum_{i=0}^{n} r_{i,f} \le R,\ \forall 1 \le f \le \omega \tag{7}$$

We model this problem as *Single-stage Inter-Coflow Scheduling Problem* (SICS).

**Theorem** 2: The SICS problem is a convex optimization problem.

*Proof:* The proof of convex optimization requires the problem has two characteristics: 1. the function is a convex; 2. the solution space is a convex set.

1. The target function $T(r)$ is convex.

Firstly, we will prove the function of $n$-dimension vector $x$, $max(x)$ is convex.

We pick two $n$-dimension vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$, $\vec{b} = (b_1, b_2, \ldots, b_n)$ and a number $t \in [0,1]$ arbitrarily. We assume $a_i = \max(\vec{a})$ and $b_j = \max(\vec{b})$. Then

$$a_k \le a_i \Rightarrow ta_k \le ta_i,\ \forall 1 \le k \le n$$

Similarly,

$$b_k \le b_j \Rightarrow (1-t)b_k \le (1-t)b_j,\ \forall 1 \le k \le n$$

As a result,

$$ta_k + (1-t)b_k \le ta_i + (1-t)b_j,\ \forall 1 \le k \le n$$

which is,

$$\max(t\vec{a} + (1-t)\vec{b}) \le t\max(\vec{a}) + (1-t)\max(\vec{b})$$

So $\max(\vec{x})$ is convex.

Also, it is obvious that the function of scalar $r_{i,f}$, $g(r_{i,f}) = \frac{V_{i,f}}{r_{i,f}}$ is convex. Together with the convexity of $\max(\vec{x})$, the function of vector $\vec{r_i}$, $h(\vec{r_i}) = \max_{f}\ g(r_{i,f})$ is convex. Then the function of matrix $r$, $T(r) = \sum_{i=1}^{n} h(r_i)$ is convex.

2. the solution space is a convex set.

The solution space is all $n \times \omega$ matrices in which every element $x$ satisfies $x \in [0, R]$. We take two matrices $A$ and $B$ arbitrarily, and then

$$\begin{cases} A_{i,f}, B_{i,f} \in [0,R], \\ \sum\limits_{i=0}^{n} A_{i,f} \le R, & (\forall 1 \le i \le n, \forall 1 \le f \le \omega) \\ \sum\limits_{i=0}^{n} B_{i,f} \le R \end{cases}$$

Clearly, $\forall\lambda \in [0,1]$, for any element $C_{i,f}$ of new matrix $C = \lambda A + (1-\lambda)B$ is

$$C_{i,f} = \lambda A_{i,f} + (1-\lambda)B_{i,f} \in [0,R]$$

Also,

$$\begin{aligned} \sum_{i=0}^{n} C_{i,f} &= \sum_{i=0}^{n}[\lambda A_{i,f} + (1-\lambda)B_{i,f}] \\ &= \lambda\sum_{i=0}^{n} A_{i,f} + (1-\lambda)\sum_{i=0}^{n} B_{i,f} \\ &\le R \end{aligned}$$

As a result, the new matrix $C$ still belongs to the solution space, so it is a convex set.

In summary, the online scheduling problem is a problem of minimizing a convex function over a convex set, so that it is a convex optimization.

□

## C. A Local Solution: IAO

Considering that the SICS problem is convex optimization, the local optimum must be the global optimum. In virtue of the interior point method [16], our algorithm Iteratively Approaching Optimal (IAO) can be constructed as Algorithm 1. As the iterative process goes on, the result gets closer to the optimum.

In Algorithm 1, $c$ is the decreasing coefficient, and its range is $[0.1, 0.5]$. $\overline{F}(r, \zeta_k)$ is a barrier function and it has an important characteristic: $\overline{F}(r, \zeta_k)$ is near to $T(r)$ if $r$ is far from the bound of solution space, while $\overline{F}(r, \zeta_k)$ is extremely large if $r$ is near the bound. With this characteristic, iteratively searching for minimizing this barrier function without constraints is bound in the solution space automatically. As the iterative process goes on, the barrier factor $\zeta_k$ decreases, and $\overline{F}(r, \zeta_k)$ would gradually approach $T(r)$, until the precision $\epsilon$ is achieved.

**Algorithm 1** Iteratively Approaching Optimal Algorithm

```
Input: data matrix $V$, initial allocation $r^{(0)}$, precision $\epsilon$
Output: rate allocating matrix $r$
 1: Initialize the barrier factor $\zeta_1 = 1$
 2: $k \leftarrow 1$
 3: while true do
 4:     Construct the barrier function $\overline{F}(r,\zeta_k) = T(r) - \zeta_k \sum_{f=1}^{\omega}(\ln(R - \sum_{i=0}^{n} r_{i,f}))$
 5:     Begin with $r^{(k-1)}$, minimize $\overline{F}(r,\zeta_k)$ without constraint, and obtain a new solution $r^{(k)}$
 6:     if $|T(r^{(k)}) - T(r^{(k-1)})| \le \epsilon$ then
 7:         break;
 8:     else
 9:         $\zeta_{k+1} \leftarrow c\zeta_k$;
10:         $k \leftarrow k + 1$;
11:     end if
12: end while
13: return $r^{(k)}$
```

In view of the high complexity of interior point method, we set the initial allocation by Weighted Fair Sharing (WFS):

$$\frac{V_{1,f}}{r_{1,f}} = \frac{V_{2,f}}{r_{2,f}} = \cdots = \frac{V_{n,f}}{r_{n,f}}\ (\forall 1 \le f \le \omega)$$

With suitable precision and initial allocation, the speed of IAO can be accelerated significantly. Because IAO focuses on the SICS problem, fairness among sub-coflows is guaranteed every time IAO is invoked. In fact, the improvement of fairness is especially evident when the bandwidth is limited for a high workload.

### *D. More Far-sighted Heuristic: MLBF*

Owing to the awareness of multi-stage, scheduler has obtained a great advance in minimizing the average CCT. However, the greedy scheduling scheme only cares about the average completion time of current sub-coflows. This might benefit the performance in the short term, but fails take account of the successive sub-coflows. Hence to some extent, they are short-sighted.

Notice that after every transmission completes, the related computation begins at once. And during this period, the sub-coflow does not need any bandwidth resource. In an extreme case, between two schedules, a short-sighted optimizer tends to schedule all the sub-coflows to transfer simultaneously, which decreases the SCCT, but the network would be completely idle during their computing. Given that the time of computation is non-negligible, considerable network bandwidth resource is wasted in these coflow's lifespan.

A far-sighted schedule is supposed to take the computation time into consideration and interlace transmissions with each other to improve the network utilization. The parallelism of computing and transferring can significantly speed up the transmission of other sub-coflows. The more scattered transmissions are, the less time they cost. In order to accomplish this target, these sub-coflows that have a less bottleneck time should complete earlier.

The bottleneck time refers to the transmission time if all the remaining bandwidth is allocated to this coflow. When remaining bandwidth is $\Lambda$ and the flow $f \in C$ transfers $D_f$ amount of data, bottleneck $\Gamma_C$ can be calculated by

$$\Gamma_C = \max_f \frac{D_f}{\Lambda_f} \tag{8}$$

We propose Multi-stage Least Bottleneck First (MLBF) heuristic. When a sub-coflow arrives or completes, this algorithm would be invoked. Firstly, it calculates the bottlenecks $\Gamma$ of sub-coflows $\mathbb{C}$, and sorts them according to $\Gamma$ in ascending order. In this order, remaining bandwidth $\Lambda$ is allocated by sub-coflows' bottlenecks, so that flows of the same sub-coflow complete at the same time. Finally, remaining bandwidth is allocated averagely.

## V. Evaluation

In our experiments, using a simulator implemented in MATLAB, we develop our testbed with these default settings: The data size of flows follows a uniform distribution in [0,200MB], there are 70 coflows in DCN and each coflow has 10 stages and 50 flows. Besides, the default computing speed is 10MB/s. The data size is large enough that every experiment will last at least 1000 simulated seconds, in order to avoid random errors.

**Algorithm 2** Multi-stage Least Bottleneck First

```
Input: sub-coflows $\mathbb{C}$, bandwidth limit $\Lambda$
Output: rate allocation $R$
 1: for all $C \in \mathbb{C}$ do
 2:     $\Gamma_C \leftarrow$ calc_bottleneck$(C, \Lambda)$ using Equation 8
 3: end for
 4: $\mathbb{C} \leftarrow$ sort_asc$(\Gamma)$ ▷ sort $\mathbb{C}$ according to $\Gamma$
 5: for all $C \in \mathbb{C}$ do
 6:     $\tau \leftarrow$ calc_bottleneck$(C, \Lambda)$
 7:     for all $f \in C$ do
 8:         $R_f \leftarrow D_f / \tau$
 9:         $\Lambda_f \leftarrow \Lambda_f - R_f$
10:     end for
11: end for
12: Allocate remaining $\Lambda$ to $\mathbb{C}$ averagely
13: return $R$
```

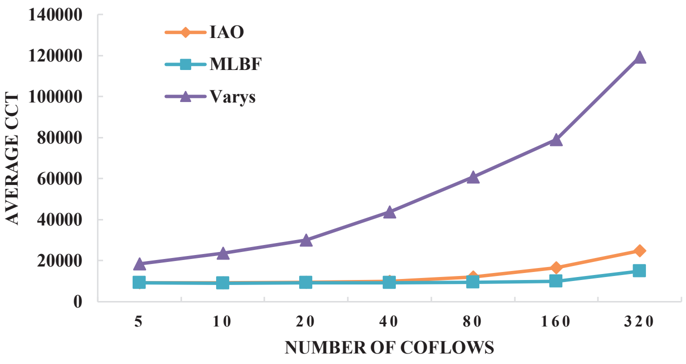

**Figure 4.** Influence of the scale of coflows.

In Figure 4, as the number of concurrent coflows in DCN increases, the average network bandwidth for each coflow is narrow, leading to an increase in average CCT of Varys. However, multi-stage aware schedulers using IAO and MLBF handle this situation well. Their improvements increase significantly with the exponential increase of coflow number. Obviously, multi-stage aware schedulers are more capable of tackling the challenge of coflows on a massive scale. In addition, when coflow number exceeds 40, the advantage of MLBF over IAO gradually appears. The improvement of IAO and MLBF over Varys increases from $2\times$ to $4.81\times$ and $2.01\times$ to $7.95\times$ respectively.

In real-world scenarios, the distribution of coflows might vary a lot. We evaluate the sensitivity of the three scheduling schemes by coflows under uniform (Uni.), exponential (Exp.), and normal (Nm.) distribution in Figure 5. We set up them by the same average volume of data. Furthermore, normal distribution is tested by setting the standard deviation 25, 50, 75, and 100. Figure 5 shows that the multi-stage aware schedulers have a steady performance while Varys fluctuates. In different distributions, MLBF and IAO still have a better performance than Varys and MLBF keeps an advantage over IAO by $1.56\times$ on average.

Computation phases account for an important part of the

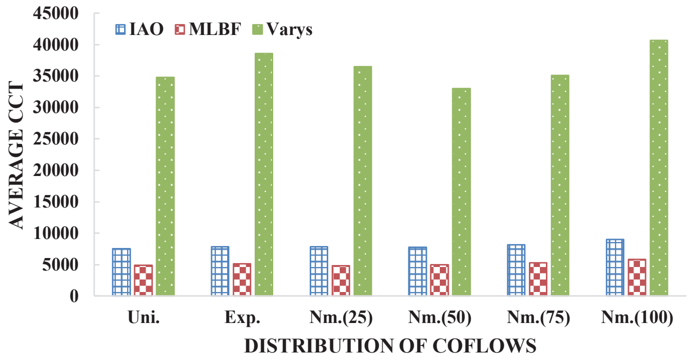

**Figure 5.** Sensitivity to the distribution of coflows.

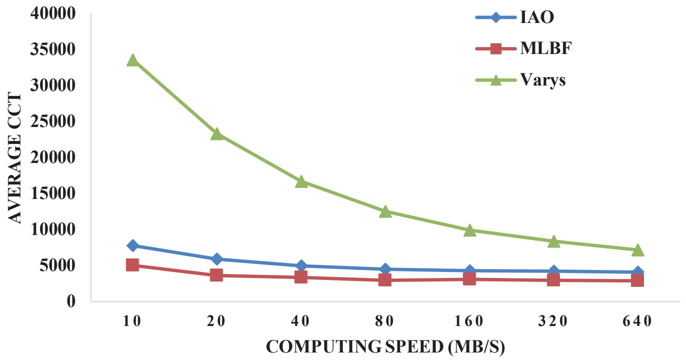

**Figure 6.** Sensitivity to the computing speed.

processing of a task. Network utilization is influenced by the computing speed because parallelism of computation and transmission can reduce the idle time of network. We test the three scheduling schemes with different computing speeds. Figure 6 shows a notable decline of average CCT of Varys with the improvement of computing speed. Varys is not aware of multi-stage characteristic, so that its CCT depends on the computation time directly and is sensitive to the computing speed. Meanwhile, IAO and MLBF leverage this characteristic and perform well - from 1.75× to 4.34× and from 2.52× to 6.74× than Varys. Owing to a better network utilization, MLBF outperforms IAO by 1.49× on average.

Besides improvement in average CCT, multi-stage awareness also brings forward the time latest coflow completes (Figure 7). We observed that the latest coflow of multi-stage aware schedulers has a much less CCT than Varys. It is because the awareness give schedulers insight to reduce unnecessary waiting time, potentially avoiding perpetual starvation.

However, IAO is more attractive under a high workload than MLBF because of its improvement of fairness among coflows. We define the improvement of average CCT (ACCT), standard deviation (STD) and the factor of workload as follows:

$$\text{Improvement of ACCT} = \frac{\text{ACCT of Varys}}{\text{ACCT of Alg.}}$$

$$\text{Improvement of STD} = \frac{\text{STD of Varys}}{\text{STD of Alg.}}$$

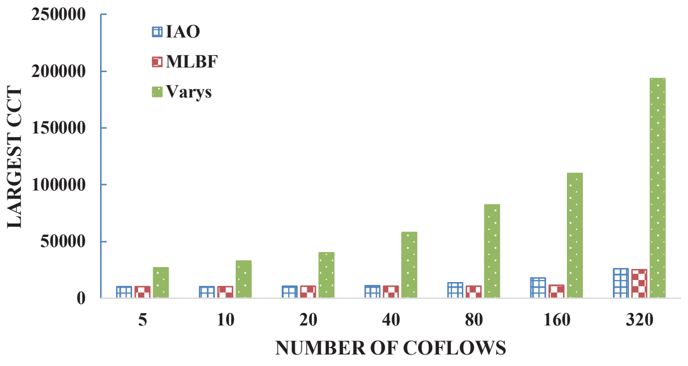

**Figure 7.** Multi-stage awareness potentially avoids perpetual starvation.

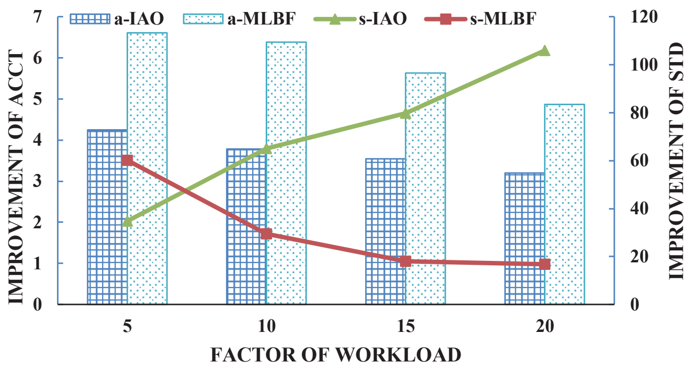

**Figure 8.** Performance of IAO and MLBF under different workloads. a- is ACCT and s- is STD.

$$\text{Factor of Workload} = \frac{\text{amount of data}}{\text{maximum of bandwidth}}$$

In Figure 8, as the workload increases, IAO has a great improvement of STD on MLBF while its performance on ACCT is slightly worse than MLBF. A less STD means a more fair scheduling, which is important to a high workload.

## VI. Discussion

**Bijective Flow** In our study, flows are assumed to be bijection. Simple though, it lays a solid foundation for our future work. We can consider a physical port as multiple virtual ports of which sum bandwidth equals to the physical port. By this way, the complex flows can be divided into many bijections, which have been studied in this paper.

**Scheduling Without Prior Knowledge** It is difficult to know the information about stages of coflows a priori. However, there still some feasible solutions. Firstly, these information can be notified when coflows register with schedulers by an API (e.g., Varys [10]). Secondly, this information can also be obtained by machine learning (e.g., CODA [19]). Although it might be not accurate enough, the applications do not need any modifications. We believe that this information is sufficient to drive a multi-stage aware scheduler well.

## VII. Related Work

**Coflow Schedulers** Coflow is recently proposed by Stoica *et al.* [9]. After that, a lot of solutions have emerged to improve coflow performance. Varys [10] is the first coflow aware scheduler which leverages the characteristics of coflow to minimize CCT. Qiu *et al.* [20] proposed a way to minimize the weighted CCT while scheduling coflows with release dates. Aalo [13] provides a scheme without prior knowledge, taking the dependency of coflows into consideration.

We focus on the multi-stage characteristic of coflow that has not been noticed before and give schedulers a fine-grained view of coflows.

**Flow Schedulers** Transport-level flow scheduling has been studied in a long time. Hedera [6] designed a dynamic scheduling system to utilize aggregate network resources. Orchestra [21] noticed the semantics between flows, and optimized at a higher level that individual flows. Baraat [22] further promoted scheduling level to task, so that schedulers become task-aware. Using expansion ratio, Zhang *et al.* [23] designed MERP to achieve efficient flow scheduling without starvation.

We schedule flows at a higher level, obtaining more semantics. Minimizing CCT is more beneficial to improve the performance of applications than minimizing FCT directly.

## VIII. Conclusion

We study the coflow scheduling and find an important characteristic that a coflow has multiple stages. In light of our observation, we formulate the *Multi-stage Inter-Coflow* problem and prove it to be NP-hard. We divide the MICS problem into many *Single-stage Inter-Coflow* problems. Then we prove the SICS problem to be a convex optimization problem. We construct a solution IAO to approach the optimal rate allocation for every SICS problem and experiments show IAO improves average CCT by up to 4.81×. However, we find this strategy is short-sighted. In virtue of SJF, we propose a far-sighted scheduling scheme – MLBF, which outperforms IAO by up to 1.65×.

## Acknowledgments

This work was supported in part by National Key R&D Program of China (2017YFB1001801), NSFC (61502224, 61472181, 61472185), CCF-Tencent Open Fund, Ministry of Education & China Mobile Research Foundation (MCM20170307), NSF (CNS 1757533, CNS 1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, and IIP 1439672), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

## References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, Jan. 2008.

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." in *Proceedings of the 2Nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2010)*.

[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys 2007)*.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proceedings of the 2013 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2013)*.

[5] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proceedings of the 2012 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2012)*.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *Proceedings of the 2010 USENIX Symposium on Networked Systems Design and Implementation (NSDI 2010)*.

[7] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: synthesizing existing transport strategies for data center networks," in *Proceedings of the 2015 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2015)*.

[8] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proceedings of the 2011 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2011)*.

[9] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNets 2012)*.

[10] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2014)*.

[11] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers." in *Proceedings of the 2015 USENIX Symposium on Networked Systems Design and Implementation (NSDI 2015)*.

[12] H. Susanto, H. Jin, and K. Chen, "Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks," in *2016 IEEE 24th International Conference on Network Protocols (ICNP 2016)*.

[13] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proceedings of the 2015 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2015)*.

[14] 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 *Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM 2015)*.

[15] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "G: Packing and dependency-aware scheduling for data-parallel clusters," in *Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2016)*.

[16] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013.

[17] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2014)*.

[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 *Proceedings of the 2009 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2009)*.

[19] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *Proceedings of the 2016 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2016)*.

[20] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the total weighted completion time of coflows in datacenter networks," in *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures (SPAA 2015)*.

[21] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proceedings of the 2011 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2011)*.

[22] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proceedings of the 2014 ACM Conference on the Special Interest Group on Data Communication (SIGCOMM 2014)*.

[23] S. Zhang, Z. Qian, H. Wu, and S. Lu, "Efficient data center flow scheduling without starvation using expansion ratio," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3157–3170, 2017.