Minimizing Average Coflow Completion Time with Decentralized Scheduling

Shouxi Luo*, Hongfang Yu*, Yangming Zhao*, Bin Wu†, Sheng Wang*, and Le Min Li*
*Key Laboratory of Optical Fiber Sensing and Communications, Ministry of Education
University of Electronic Science and Technology of China, Chengdu, P. R. China
†School of Computer Science and Technology, Tianjin University, Tianjin, P. R. China

***Abstract*—In current data centers, an application (e.g. MapReduce) usually generates a collection of parallel flows sharing a common goal. These flows compose a *coflow* and only completing them all is meaningful. Accordingly, minimizing the average coflow completion time (CCT) becomes a critical objective for flow scheduling. In this topic, the state-of-the-art centralized method, Varys, achieves a good average CCT; but it has the scalability problem. Alternatively, the only existing decentralized method, Baraat, suffers from the head-of-line blocking problem. To solve these problems, we propose D-CAS, a *preemptive*, *decentralized*, *coflow-aware* scheduling system in this paper. D-CAS pursues coflow-level minimum-remaining-time-first (MRTF) principle by leveraging a simple negotiation mechanism between each coflow's data senders and receivers. As the MRTF principle is inherently preemptive and proven to be a near-optimal guideline to minimize average CCT, D-CAS avoids the head-of-line blocking problem and gets good performances. Through extensive simulations, we find that D-CAS achieves a performance close to Varys (gap $< 15\%$) and outperforms Baraat significantly (about 1.4–$4\times$).**

I. Introduction

Today's data centers widely employ cluster computation frameworks (e.g. MapReduce, Dryad, CIEL, and Spark) to deal with the increasing outsourcing demands. In these frameworks, data-intensive jobs are divided into multiple successive data-parallel computation stages; and a succeeding computation stage cannot start until getting *all* its required inputs, which is exactly the outputs of the previous stage. Furthermore, the transmission of the intermediate data is not a negligible phase in a job [1]–[3]. For example, some real traces from Facebook show that, the data transferring phase between successive stages accounts for 33% of the running times of jobs in the system [1]. Accordingly, speed up the data transfer between computation stages will accelerate the job completion and increase the data center utilization [1]–[3].

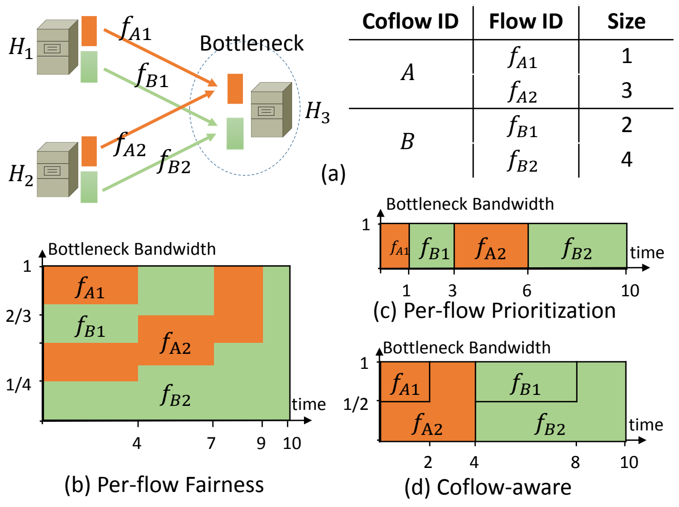
The data transfer between successive stages, which often involves a group of parallel flows, completes only when all its flows finish. This makes the flow scheduling in data center networks (DCNs) quite challenging since the semantics among these flows needs to be considered. In this paper, we are to speed up the completion of such data transfers coupled with the concept of *coflow* [4], which is defined as *a collection of flows that share a common performance goal*, e.g., minimizing the completion time of the latest flow. Then, how to schedule flows to minimize the average coflow completion time (CCT) is the objective we pursue in the paper.

This work is supported in part by the 973 Program under Grant No. 2013CB329103, the 863 Program under Grant No. 2015AA011901, the National Natural Science Foundation of China under Grant No. 61271171, the Open Foundation of State Key Laboratory of Networking and Switching Technology under Grant No. SKLNST-2014-1-09, and Huawei Research Funding under Grant No. YB2013120161.

Many existing works [1]–[3] focus on minimizing average CCT in DCNs. To the best of our knowledge, Varys [2] and Baraat [3] are the state-of-the-art schemes in centralized and decentralized manner, respectively. However, centralized schemes like Varys have the scalability problem. On one hand, it is difficult for the central controller to collect all the real-time coflow information in a large scale network; on the other, calculating scheduling schemes for the entire network involves large-scale computing tasks. Moreover, as the scheduling schemes need be executed by data senders, it's almost impossible to accurately synchronize them in real time. Different from Varys, Baraat [3] is a FIFO-based decentralized coflow scheduler which performs well in networks with homogeneous coflows. Due to its *non-preemptive* schedule policy, Baraat endures the *head-of-line blocking problem* [2]. Despite detecting large size coflows online and using fair sharing to mitigate head-of-line blockings, it still gets bad performances (even worse than the naive per-flow fair sharing) when coflows are heterogeneous (detailed in Section II-C and Section V).

Motivated by above discussions, we design D-CAS, a *preemptive*, *Decentralized*, *Coflow-Aware* Scheduling system, to minimize the average CCT. D-CAS schedules flows by dynamically setting priorities to their packets. It is proved that *minimum-remaining-time-first* (MRTF, i.e. scheduling the coflow with the smallest ideally completion time first) is a near-optimal guideline to pursue the minimum average CCT [2] (namely SEBF in [2]). Accordingly, we design a local negotiation mechanism to derive flow priorities in D-CAS. In the negotiation procedure, each coflow's data senders periodically announce their desired-priorities to the data receivers based on their remaining coflow size. For the received desired-priorities, the receiver gets feedback to senders to negotiate their flow priorities. At last, every sender determines the updated priority based on these feedbacks and its local information.

We implement a simulator (see Section V) to evaluate the performance of D-CAS. Extensive simulations imply: D-CAS improves the average CCT about 2–$3\times$ over per-flow fairness; the performance improvement is close (gap $< 15\%$) to that of the state-of-the-art centralized scheme Varys, and outperforms



Coflow ID	Flow ID	Size
A	f_{A1}	1
	f_{A2}	3
B	f_{B1}	2
	f_{B2}	4

Fig. 1. Motivating Example. (a) Four concurrent flows from two coflows competing for a single bottleneck link; (b) Fair sharing, $avg(CCT)$=9.5; (c) Smallest-flow-first, $avg(CCT)$=8; (d) Smallest-coflow-first, $avg(CCT)$=7.

the only existing decentralized scheme Baraat by 1.4–4×.

Roadmap We first briefly show the background and motivation of our work in Section II. In Section III, we discuss the design of D-CAS and give an overview, followed by system details in Section IV. Extensive simulations are presented in Section V and we conclude the paper in Section VI.

II. Motivation

In this section, we present three key desirable properties for minimizing average CCT in DCNs, which motivate our design of D-CAS.

A. Why Coflow-aware Scheduling

Recall that the goal our scheduler pursues is to minimize the average CCT, which is limited by the flow completing last. So, conventional coflow-agnostic scheduling methods designed to minimize the average flow completion time (FCT) (e.g. pFabric [5] and PDQ [6]) cannot get the optimum solution.

Take the case shown in Fig. 1 as an example. There are four concurrent flows belonging to two coflows (Coflow A: f_{A1}, f_{A2}; Coflow B: f_{B1}, f_{B2}.). These flows arrive simultaneously and their demands are shown in Fig. 1a. With per-flow fair sharing scheme, the scheduling result is shown in Fig.1b. In this case, all the unfinished flows in the network obtain the same bandwidth, and flows $[f_{A1}, f_{A2}, f_{B1}, f_{B2}]$ will complete at time [4, 9, 7, 10]. Hereby, coflow A and coflow B complete at time 9 and 10, respectively. Their average CCT is $\frac{9+10}{2} = 9.5$. Similarly, the smallest-flow-first scheme that minimizes average FCT can only derive the average CCT $\frac{6+10}{2} = 8$ as Fig.1c shows. If the coflow-aware flow scheduling scheme, such as *smallest-coflow-first* (i.e. the coflow with the minimum total flow volume is scheduled first) is adopted, the average CCT is only $\frac{4+10}{2} = 7$ as shown in Fig.1d. There is a saving of $\sim$ 26% compared to fair sharing scheme and a saving of $\sim$ 13% compared to flow level scheduling scheme.

In a word, *the coflow-aware scheduling policy can bring benefits to the average CCT in DCNs.*

B. Why Decentralized Scheduling

Intuitively, if all the coflow information is available, we can schedule all the coflows following the MRTF policy, which is proved to be the best scheduling principle to pursue the average CCT [2].

However, above thought is not practical in real settings. At first, current data centers have hundreds of thousands of hosts and millions of concurrent flows [7], it is difficult to collect all flow information (such as the sources, destinations and remaining size), to calculate a global scheduling scheme, and to enforce it to all flows in real time. Accordingly, the delay of the centralized scheduling is too long for the pervasive small coflows since these coflows may complete within a few RTTs [3]. Second, the centralized scheduling system also have other problems such as fault-tolerance.

Accordingly, *the decentralized scheduling is preferred in a system to improve the average CCT in DCNs.*

C. Why Preemptive Scheduling

To minimize the average CCT in the network, we should pursue the MRTF principle in a decentralized manner. In other words, on every host, the smaller coflows should be scheduled before the larger ones. In an online system, we cannot suppose the smaller coflows would arrive earlier than the larger ones. Therefore, the preemptive scheduling scheme is necessary for D-CAS. Otherwise, the system may suffer from the head-of-line problem, i.e. small coflows arriving later being blocked by the early large coflows.

Take the only existing decentralized coflow-aware (namely task-aware in [3]) scheduling system, Baraat, as an example. Baraat is a non-preemptive system and solves the head-of-line blocking problem by deploying a limited multiplexing (LM) scheme on switches. LM would dynamically change the level of multiplexing and let the lower priority flows to be served when the current coflow is detected as large. Due to its non-preemptive property, there are two major shortcomings in Baraat. First, LM scheme performs badly when the coflow sizes are heterogeneous. In some cases, it may be even worse than the naive fair sharing scheme (see the simulation results in Fig. 3). Second, with the increasing of multiplexing level, the performance of Baraat is approaching that of fair sharing scheme, which is demonstrated to be unsuitable for minimizing average completion time [2, 5, 6].

Hereby, *preemption is a necessary property to minimize the average CCT in DCNs.*

III. D-CAS Overview

Motivated by above discussions, we design D-CAS, a *coflow-aware*, *decentralized*, *preemptive*, and *starvation-free* system, to minimize the average CCT for data-incentive DCNs. In D-CAS, each source host (i.e. the data sender) determines the priority of its flows based on its local information and feedbacks from the receivers. To clearly present D-CAS, we first give some key definitions in Section III-A, then show the key idea of our design in Section III-B. In the end, we show the overall framework of D-CAS in Section III-C.

A. Key Definitions

Coflow: It is a set of flows that are for the same purpose. We say a coflow is the *parent coflow* of all its flows. The *length* of a coflow is defined as the volume of its largest flow, while the *width* is the number of flows in it. By summing up the volume of all its flows, we get the *size* of this coflow.

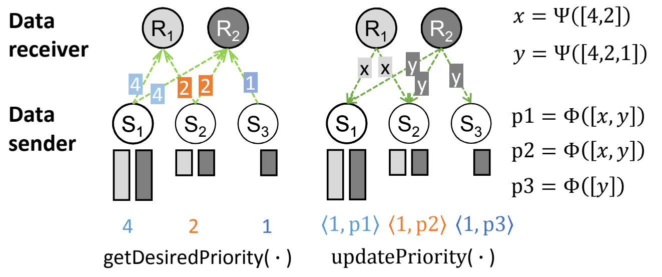
Subcoflow: A subcoflow S consists of all the flows in the same coflow C that stem from the same *source host*. For simplicity, we call C is the *parent coflow* of subcoflow S. Thus, a subcoflow can be identified by the tuple of its parent coflow and source host. Similarly, the *size* of a subcoflow is the volume of all its flows.

Priority: In D-CAS, each host dynamically sets priorities to packets to realize flow scheduling. As D-CAS schedules small coflows before large coflows, we design the priority number as a tuple $\langle T, P\rangle$. We call the two items, T and P, as *main priority* and *secondary priority*, respectively. Also, we say, $\langle T_1, P_1\rangle$ is a higher priority than $\langle T_2, P_2\rangle$, iff $T_1 < T_2$ or $T_1 = T_2$ and $P_1 < P_2$.

B. Basic Ideas for Coflow-aware Scheduling

Though it is NP-hard to get the minimum average CCT for a given set of coflows, MRTF is still a good guideline for our scheduling since it will lead to a near-optimal solution [2]. In a decentralized scenario, each host only has the information of its own flows; it does not have the global view of coflows or the entire network. Therefore, each host can only schedule its flows following the "subcoflow-level" minimum-remaining-time-first (SL-MRTF) principle to approach MRTF. Following SL-MRTF, all the flows in a subcoflow get the same priority. Each packet carries its priority number set independently by its subcoflow sender, and switches send the packet with the highest priority when transmitting (similar to pFabric [5]).

Ideally, the SL-MRTF scheme can get a good performance if every coflow only has exactly one subcoflow (i.e. exactly follows the MRTF principle). Unfortunately this is not the case in current DCNs, where flows in each coflow are originated from different hosts, and hence involve multiple subcoflows. Once these multiple subcoflows have different remaining sizes, SL-MRTF may degrade since it is not exactly following the "coflow-level" MRTF principle. Revisit the case shown in Fig.1 as an example. In this case, both A and B contains two single-flow subcoflows; SL-MRTF degenerates to smallest-flow-first and becomes coflow-agnostic. To alleviate this effect, we introduce a feedback scheme into D-CAS. It works as follows: for each subcoflow, its data sender announces a *desired-priority* presenting its ideally remaining time to all the subcoflow's data receivers; when receiving a *desired-priority* message, the data receiver uses all the received *desired-priorities* from the same coflow to compute a *feedback-priority*, and sends it back to the announcer. Following this, the subparts of a coflow (i.e. subcoflows) can negotiate their priorities without any third-party controller. When all receivers choose the lowest priority to relay, D-CAS approaches "coflow-level" MRTF in a decentralized manner.



(a) Announce *desired-priority* (b) Get priority feedbacks then update

Fig. 2. Framework overview: how a coflow's subcoflows (S_1, S_2, S_3) negotiate with the help of its own data receivers (R_1, R_2).

In addition, when a subcoflow is smaller than a predefined threshold, we do not schedule it according to the SL-MRTF principle any more. In this case, we believe FIFO is a better choice since the feedback delay is too long for these subflows that may complete in a few RTTs. On the other hand, the simulation of [3] also shows, the FIFO policy can achieve a good performance for minimizing the average CCT when the flow size (i.e. coflow *length*) is distributed in a small range.

C. D-CAS *in a Nutshell*

Based on the analysis in previous subsection, the design of D-CAS can be summarized as follows: if a subcoflow's remaining volume is smaller than a threshold, schedule it using FIFO policy like Baraat; otherwise, schedule its flows using the negotiated priority. We design the control plane for priority negotiating as Fig.2 shows. Note that, the shown control procedure is just for a single coflow. That is to say, each host will launch such a procedure for every subcoflow to determine its flow priority. For each data sender, it periodically announces the *desired-priority* to the data receivers based on the subcoflow's remaining size. See the example in Fig.2(a), the *desired-priorities* of the three data senders are 4, 2 and 1, respectively, based on their remaining sizes.

On getting the *desired-priority* message, each data receiver replies a feedback that is derived by a predefined function $\Psi(\cdot)$ and the received messages. As shown in Fig.2(b), R_1 and R_2 get feedbacks to the senders with $x = \Psi([4, 2])$ and $y = \Psi([4, 2, 1])$, respectively. At last, when the data sender collects feedbacks from data receivers, it gets the negotiated priority by computing $\Phi([x, y])$. Then the sender updates the subcoflow's priority based on the negotiated priority or some local schemes like preventing starvation.

All relevant technical details such as the design of function $\Psi(\cdot)$ and $\Phi(\cdot)$ will be discussed in the following section.

IV. Design Details

In this section, we show what operations each sender and receiver should perform in detail, and discuss how to design the functions that D-CAS uses to determine the flow priority.

Sender In D-CAS, each sender periodically (δ-interval) detects subcoflow information and updates data transfers' priorities, as Pseudocode 1 shows. For each subcoflow S whose parent coflow is c, if its remaining size (rem in Line 4) on the sender is less than the predefined $thresholdVolume$

(e.g. set it as BDP, i.e. bandwidth delay production), its flows will be sent with priority value $\langle 0, cp \rangle$, where cp is the parent coflow's ID representing its arrival order (same to the Task-ID in [3]). Otherwise, S is identified as a large subcoflow and the main priority of its flows should be set to 1. If this subcoflow has not received any service during the last T-interval ($T \gg \delta$), its secondary priority will be set to be 0 to prevent starvation (Line 8). In this case, all the un-served subcoflows share the network using per-flow fairness mechanism. If not, the priority negotiation is launched for the secondary priority. To this end, each sender announces its desired priority to the receivers and waits several RTTs (Line 11). Regularly, the sender uses the received feedbacks to calculate the secondary priority with $\Psi(\cdot)$ (Line 15). However, if all the feedbacks get lost due to the network congestion or failure, the sender just sets the secondary priority to its desired priority (Line 17).

Pseudocode 1 Coflow-aware Flow Scheduling in Sender

```
1: procedure SCHEDULE(Subcoflows $\mathbb{S}$)          ▷ recall every-$\delta$
2:     for all $S \in \mathbb{S}$ do
3:         $c \leftarrow S.coflowID$
4:         $rem \leftarrow getLocalRemSize(S)$
5:         $d \leftarrow getDesiredPriority(S)$
6:         if $rem < thresholdVolume$ then
7:             $p \leftarrow \langle 0, getCoflowPriorityValue(c) \rangle$   ▷ FIFO
8:         else if $S.waitTime() > T$ then
9:             $p \leftarrow \langle 1, 0 \rangle$                    ▷ Starvation-free
10:        else
11:            Announce $d$ to all $S$'s data receivers.
12:            Waiting several RTTs.
13:            $M \leftarrow getFeedbackPrioritySet(c)$
14:            if $M$ is not empty then
15:                $p \leftarrow \langle 1, \Phi(M) \rangle$
16:            else
17:                $p \leftarrow \langle 1, d \rangle$
18:            end if
19:        end if
20:        Update the priority of $S$'s data transfers to $p$.
21:    end for
22: end procedure
```

Receiver In D-CAS, each receiver maintains a cache for the latest desired-priority of each subcoflow. When receiving a desired-priority, it operations as Pseudocode 2 shows: (i) get (Line 3) and update the cache (Line 4); (ii) derive the feedback using its cached desired-priority information from the same coflow (Line 5), and (iii) send it back (Line 6).

Design of $\Psi(\cdot)$ and $\Phi(\cdot)$ It is obvious that the system effectiveness is heavily impacted by the two functions, $\Psi(\cdot)$ and $\Phi(\cdot)$. They are used to calculate the feedbacks by the receivers and to calculate the new priority by the senders, respectively. As to pursue the MRTF, the subcoflow with the larger remaining time should be set a lower priority (i.e. a larger priority value). Hereby, when a receiver gets multiple desired-priorities, the subcoflow with the largest remaining time is more likely to be

Pseudocode 2 Reply Feedbacks in Receiver

```
1: procedure REPLY(msg $m$)       ▷ message from the sender
       ▷ $m$ stores the subcoflow's desired priority and information
2:     $c \leftarrow m.coflowID$
3:     $B \leftarrow getCachedMsgs(c)$
4:     Update $B$ using $m$ and remove expired messages.
5:     $p \leftarrow \Psi(B.desiredPriorityValues())$
6:     Send the feedback-priority of $c$ ($p$) to $m.src$.
7: end procedure
```

the bottleneck of their parent coflow. Therefore, receiver can simply send back the largest priority value to the senders as the feedback. Formally, $\Psi([p_1, p_2, \dots]) = max(p_1, p_2, \dots)$. For the similar reason, $\Phi(\cdot)$ should also return the largest received priority value for each sender.

About $getDesiredPriority(\cdot)$ This function is used to derive the desired priority of each subcoflow by every sender. To pursue the MRTF principle, the subcoflow with larger remaining completion time should have the larger priority value. Therefore, the senders in D-CAS heuristically set this value as $\frac{RemainingSubcoflowSize}{NICLineRate}$, i.e. the time to complete the entire subcoflow if it occupies all the bandwidth.

It is worth noting that D-CAS can also be changed to be other existing system by adopting different $\Psi(\cdot)$, $\Phi(\cdot)$, and $getDesiredPriority(\cdot)$. For example, if we set $getDesiredPriority(\cdot) \equiv 1$ and $\Psi(\cdot) = \Phi(\cdot) = max(\cdot)$, D-CAS degenerates to be a per-flow fair-sharing system.

V. Simulation

We implement a Python-based simulator to evaluate the performance of D-CAS by comparing it with Varys, Baraat, and the per-flow fairness mechanism. Our simulator shares the similar design with that of Varys [8], and performs a detailed replay of the similar coflow traces as well. Extensive simulation results demonstrate: 1) D-CAS improves the average CCT about 2–$3\times$ over per-flow fairness; 2) D-CAS achieves a performance very close to Varys–the performance gap between D-CAS and Varys is less than 15%; 3) D-CAS outperforms Baraat by about $4\times$ when coflows are heterogeneous, and about $1.4\times$ when coflows are homogeneous.

A. Methodology

Setup The coflow traces we used are synthesized with the same trace generator as Varys [2, 8]. Similarly to the setting of Varys, all the coflows are categorized to be four types in terms of their length (the size of the largest flow in bytes for a coflow) and width (the number of parallel flows in a coflow): *Narrow&Short*, *Narrow&Long*, *Wide&Short* and *Wide&Long*, where a coflow is considered to be *short* if its length is less than 10MB, and *narrow* if it involves at most 50 flows.

By defaults, coflows are assumed to arrive in a *Poisson* process with parameter λ, and each above type of coflows constitutes 52%, 16%, 15%, 17% of the coflow stream, respectively (according to the statistics reported by [2]). Without declaration, simulation results are based on 400 coflows served by an 80-hosts cluster with following settings:

1). All the flows in a coflow arrive at the same time [1, 2], and the upper bound of flow volume (i.e. the upper bound of coflow length) is 500 MB.
2). The network model (i.e. the entire datacenter fabric) is abstracted out as one non-blocking switch [2, 5] interconnecting all the machines, and we only focus on its ingress and egress ports (e.g., machine NICs); both the ingress and egress are set with the capacity of 1 Gbps.
3). The arrival rate of coflow is set to $\lambda = \frac{NetworkThroughput}{MeanOfCoflowSize}$, this is to make the network neither be overloaded or underloaded since $E(NetworkLoad) = \frac{\lambda \times MeanOfCoflowSize}{NetworkThroughput} = 1$ with such a setting.
4). For D-CAS, we use $T = 1$ second, $\delta = 100$ milliseconds, $expiredTime = 2$ seconds, and $thresholdVolume = 1$ MB.
5). For Varys, since it only reschedules flows when a coflow arrives and completes, it cannot fully utilize the network resource [2]. To solve this, we let Varys reschedule flows on flow arrival and completion in our simulation.
6). For Baraat, we set its threshold of large-coflow identifying to be 80th percentile of the coflow size. In fact, we repeat the tests multi-times and observe that the results are insensitive to the threshold setting in our simulation.

Metrics We use the improvement in average CCT as our primary metric and the improvement factor is defined as

$$\text{Factor of Improvement} = \frac{\text{Current Duration}}{\text{Modified Duration}}.$$

In all tests, we use the duration of per-flow fair sharing as a baseline, since per-flow fair sharing represents the operation of current transport protocols (e.g. TCP, DCTCP) in DCNs.

B. Results

We investigate the performance of D-CAS under different *network load*s, *cluster/network scale*s and *coflow type*s.

The Performance of D-CAS Fig. 3 shows the detailed improvements of Baraat, D-CAS, and Varys w.r.t. per-flow fair-sharing, respectively. It implies that both D-CAS and Varys greatly reduce the average and 95th percentile CCT across all coflow types. For example, the improvement factors of D-CAS on average CCTs are 38.502 (Narrow&Short), 26.162 (Narrow&Long), 9.625 (Wide&Short), 2.145 (Wide&Long) and 3.036 (ALL), while that of Varys are 39.278 (Narrow&Short), 30.578 (Narrow&Long), 9.972 (Wide&Short), 2.363 (Wide&Long) and 3.361 (ALL). We note that, the performances of D-CAS and Varys are very close, and the gap between their improvements is less than 10%. This is due to the fact that the simple negotiation mechanism in D-CAS helps each coflow using its maximum-subcoflow-remaining-size[1] as its priority. Thus the coflow with the smaller remaining size on all its senders would be more likely to use the network. This makes the scheduling scheme in D-CAS approach the MRTF policy.

[1]When all hosts have the same $NICLineRates$, $getDesiredPriority(\cdot)$ is equivalent to $getLocalRemSize(\cdot)$.

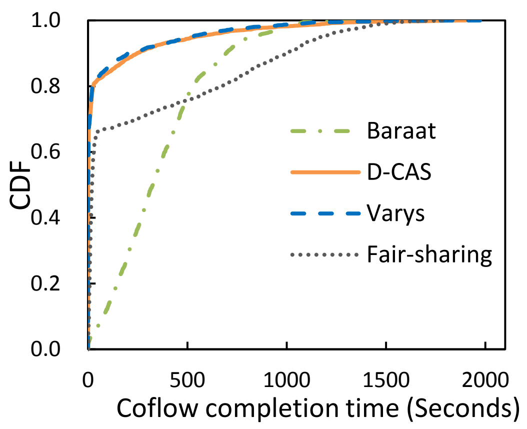


Fig. 4. CCT distributions for different scheduling schemes in the simulation.

The results also show the bad performance of Baraat on scheduling heterogeneous coflows. Baraat only speeds up the completion of long (both *Narrow&Long* and *Wide&Long*) coflows a little, while degrading the completion of short coflows. Worsely, it is inferior to the baseline on the overall average CCT. This is because, the FIFO policy Baraat using leads to serious head-of-line blocking problem and its limited multiplexing (LM) scheme cannot solve this problem absolutely, especially when facing heterogeneous coflows.

Fig. 4 shows the CDFs of CCT under different scheduling schemes. It implies that, though the preemptive schemes like D-CAS and Varys reduce the average CCT, they prolong the CCT tails. Conversely, non-preemptive schemes like Baraat can cut the long tails of CCT, but enlarge the average CCT.

Impact of Network Load To study the impact of network load ($E(NetworkLoad) = \frac{\lambda \times MeanOfCoflowSize}{NetworkThroughput}$), we vary the arrival rate of coflows and investigate the performance of different scheduling scheme. Fig.5(a) shows these simulation results. It indicates: 1) regardless of the network load, the improvement factor of D-CAS is always close to that of Varys and outperforms Baraat a lot; 2) the improvement factors of D-CAS and Varys slightly increase with the network load, while that of Baraat is stable. This is because, the heavier the network load is, the larger optimization space there is for preemptive scheduling schemes.

Impact of Cluster/Network Scale To explore the impact of cluster/network scale, we investigate their improvement factors under different sizes of clusters. The result in Fig.5(b) shows both the improvement factors of D-CAS and Varys grow with the network size (coincides with the simulation results in Varys [2]), while that of Baraat is relatively stable. Such a phenomenon is mainly caused by the setting that we always adjust λ to make $E(NetworkLoad) = 1$ in simulations. Under such a setting, the $NetworkThroughtput$ would grow linearly with the cluster size and there will be more concurrent coflows in a larger cluster. Accordingly, preemptive coflow-aware scheduling methods like D-CAS and Varys get more optimization spaces than non-preemptive methods like per-flow fairness and Baraat.

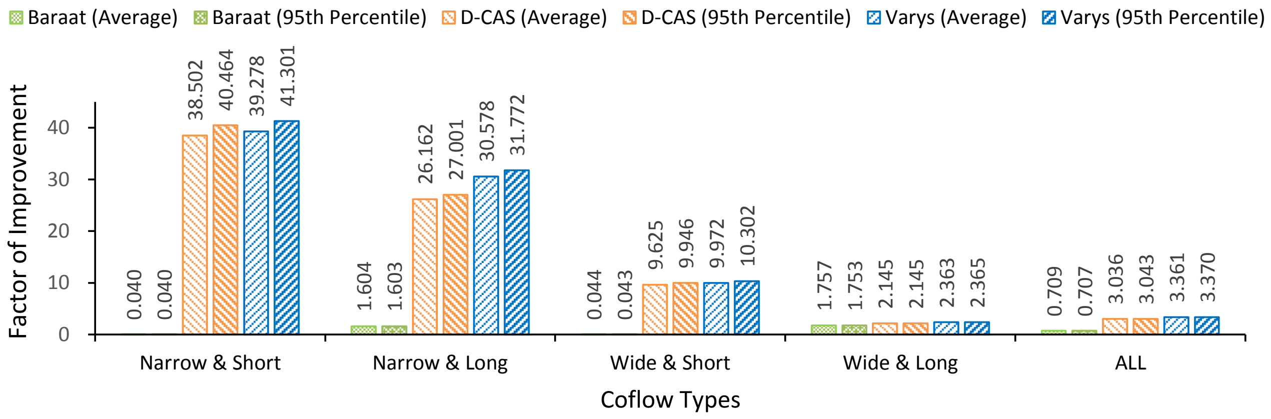
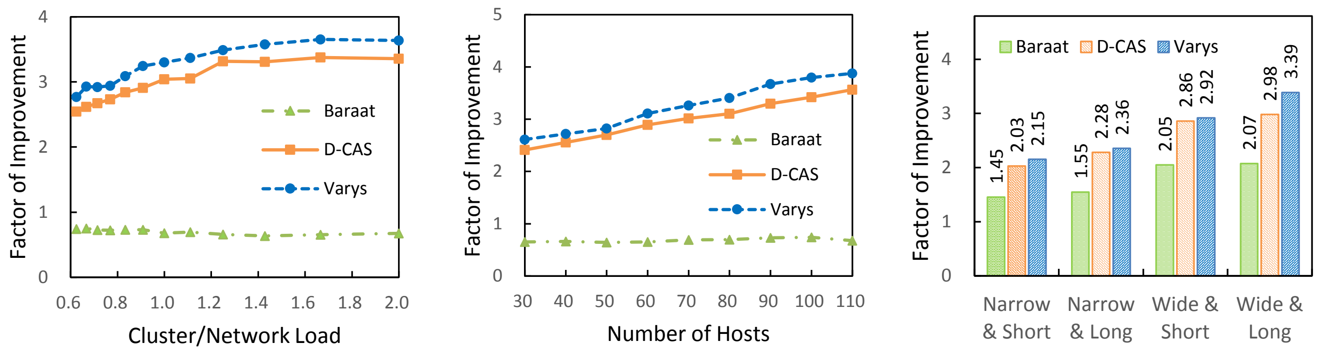


Fig. 3. Improvements of Baraat, D-CAS, and Varys w.r.t. the default per-flow fairness mechanism in the average and 95th percentile.



(a) Impact of network load

(b) Impact of cluster/network scale

(c) Impact of coflow type

Fig. 5. Improvements in the average CCT under different settings. Note that all the factor of improvements use per-flow fairness as baseline.

Impact of Coflow Type: We now study the impact of coflow types. To highlight the comparison, we investigate the performance of different scheduling schemes when there is only one type of coflows in the network. In each simulation, we also hold $E(NetworkLoad) = 1$. From the results shown in Fig.5(c), we make two important observations. First, Baraat outperforms per-flow fairness in all the four cases. This is because the FIFO scheduling policy gets a good performance when coflows are homogeneous. Second, all three coflow-aware scheduling schemes perform better when the coflow is longer and wider, and their increments of improvement factors are more sensitive to coflow *width* than coflow *length*. This is due to the fact that, the larger *width* coflows have, the more likely they may interleave with each other, in which condition, the performance of coflow-agnostic per-flow fair sharing mechanism falls increasingly further behind as it is coflow-agnostic.

Besides, we also observe that: 1) D-CAS always outperforms Baraat, about $4\times$ when coflows are heterogeneous, and about $1.4\times$ when coflows are homogeneous; 2) D-CAS achieves a performance very close to that of Varys, their performance gap is always less than 15% in simulations.

VI. Conclusion

We proposed a decentralized flow scheduling system - D-CAS to minimize the average CCT in DCNs. Different from the state-of-the-art decentralized scheduling system Baraat, which is a non-preemptive system and pursues the FIFO-based policy, D-CAS is a preemptive system and pursues the MRTF principle. Accordingly, it avoids the head-of-line blocking problem and achieves a much smaller average CCT. Our numerical results demonstrated the superior performance of D-CAS over Baraat and is close to the state-of-the-art centralized scheduling method Varys.

References

[1] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, “Managing data transfers in computer clusters with orchestra.” in *Proc. ACM SIGCOMM*, 2011, pp. 98–109.

[2] M. Chowdhury, Y. Zhong, and I. Stoica, “Efficient coflow scheduling with varys,” in *Proc. ACM SIGCOMM*, 2014, pp. 443–454.

[3] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, “Decentralized task-aware scheduling for data center networks,” in *Proc. ACM SIGCOMM*, 2014, pp. 431–442.

[4] 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)*. ACM, 2012, pp. 31–36.

[5] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, “pfabric: Minimal near-optimal datacenter transport,” in *Proc. ACM SIGCOMM*, 2013, pp. 435–446.

[6] C.-Y. Hong, M. Caesar, and P. B. Godfrey, “Finishing flows quickly with preemptive scheduling,” in *Proc. ACM SIGCOMM*, 2012, pp. 127–138.

[7] 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. ACM SIGCOMM*, 2009, pp. 51–62.

[8] M. Chowdhury, “Flow-level simulator for coflow scheduling used in varys,” https://github.com/coflow/coflowsim.