Leveraging Multiple Coflow Attributes for Information-Agnostic Coflow Scheduling

Shuo Wang*, Jiao Zhang*,†, Tao Huang*†, Tian Pan*, Jiang Liu* and Yunjie Liu*†
{shuowang, jiaozhang, htao, pan, liujiang, yunjieliu}@bupt.edu.cn
*State Key Laboratory of Networking and Switching Technology, BUPT, China
†Beijing Advanced Innovation Center for Future Internet Technology, Beijing, China

***Abstract*—Recently, designing information-agnostic coflow scheduling mechanisms attracts much attention since by leveraging priority queues, they could reduce coflow completion time in data-parallel clusters without a priori knowledge, such as flow size, coflow size. However, existing information-agnostic mechanisms generally schedule coflows only according to the sent data size of different coflows and ignore other useful coflow-level attributes like width, length and communication patterns.**

In this paper, we investigate that the coflow completion time could be further decreased by jointly leveraging multiple coflow-level attributes. Based on this investigation, we present a Multiple-attributes-based Coflow Scheduling (MCS) mechanism to reduce the coflow completion time. In MCS, a *Shortest and Narrowest Coflow First* (SNCF) algorithm is designed to separate coflows based on their widths and estimated lengths at the start of a coflow. During the transmission of coflows, one type of demotion thresholds employed in previous coflow scheduling mechanisms is too crude for various coflows. Therefore, we proposed a *double-threshold* scheme to adjust the priorities of narrow (small coflow width) and wide (large coflow width) coflows according to different thresholds. Trace-driven simulations with production workloads show that MCS outperforms the previous information-agnostic scheduler Aalo, and reduces the coflow completion time of small coflows.

***Index Terms*—Coflow; Datacenter network; Scheduling;**

I. Introduction

Data-parallel cluster computation frameworks (e.g., MapReduce [1], Dryad [2], and Spark [3]) are widely used to analyze big data in data centers. In these frameworks, data-parallel applications usually have multiple successive computation stages, and a succeeding computation stage can start only after all the intermediate data generated by the previous stage are in place [4]. Generally, the intermediate data transfer involves a collection of flows between two groups of machines, and the collection of flows is defined as a coflow [5]. For example, a typical coflow is the shuffle between the mappers and reducers in MapReduce. Recent work has shown that the intermediate data transmission accounts for more than 50% of applications' completion times [6], and scheduling flows at coflow-level can significantly reduce the completion times of the communication stages [4], [7]–[10].

This work is supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61502049, National High-Tech Research and Development Plan of China (863 Plan) under Grant No. 2015AA016101, Beijing New-star Plan of Science and Technology under Grant No. Z151100000315078, and Young talent development program of the CCF and CAST.

The coflow scheduling mechanisms could be broadly classified into two categories: information-aware coflow scheduling and information-agnostic coflow scheduling. Information-aware coflow scheduling assumes the coflows' information could be know priori [4], [7]–[9], [11], [12]. However, identifying coflows and knowing all coflows' characteristics are infeasible in the cloud data centers. On the one hand, most of the proposed information-aware coflow schedulers have their own coflow API, and it is hard to correctly modify existing data-parallel applications to use the API, making it difficult to identify coflows prior. On the other hand, data is usually transferred as soon as it is generated in data-parallel applications [10], making it hard to know coflow's characteristics, such as coflow size.

To overcome the drawbacks of information-aware coflow scheduling mechanisms, information-agnostic coflow scheduling mechanisms have been proposed to schedule coflows without the prior knowledge of coflow characteristics [10], [13], [14]. At the heart of these mechanisms is a sent-bytes-based rate allocation. More specifically, they place coflows into priority queues, and a coflow is gradually demoted from the highest priority queue to lower priority queues when its sent bytes exceeds predefined thresholds. However, through our observation, we find that 1) using sent bytes cannot separate small and large coflows when they arrive in batch, which causes Head-of-line (HOL) blocking and thus increases the completion time of small coflows, 2) using sent bytes is too crude to adjust priorities, which may increase the completion time of coflows.

The above shortcomings cause us to conclude that the design space for information-agnostic scheduling should not be limited to the sent-bytes-based rate allocation. Actually, we note that coflows have many other useful attributes, such as coflow width, coflow length and communication patterns. Unfortunately, these useful attributes are ignored by existing information-agnostic coflow scheduling algorithms, and whether jointly leveraging these attributes can further reduce the completion times of coflows remains unexplored.

In this paper, we first investigate that jointly leveraging multiple coflow-level attributes has the potential to largely decrease the coflow completion time. Then, based on this investigation, we propose MCS, a simple yet effective solution that leverages multiple coflow attributes. First, MCS presents *Narrowest and Shortest Coflow First* to place newly-

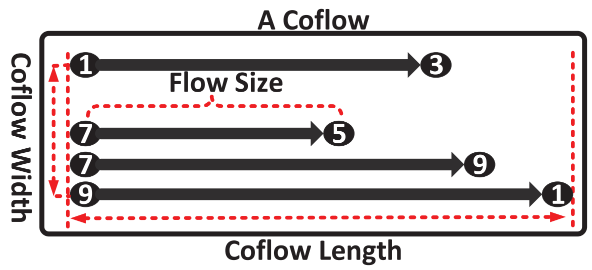


Fig. 1: The attributes of coflows

arrived coflows into different priority queues. This algorithm effectively avoids the HOL blocking problem and reduces the completion time of small coflows. Second, to mimic *Least Attained Service (LAS)* heuristic algorithm [10], [15], the priorities of coflows should be dynamically adjusted during their transmission. Previous solutions only use one type of demotion threshold to adjust the priorities for all the coflows, which is too crude. MCS solves this problem by using two types of thresholds and presents a *double-thresholds scheme*. In the *double-threshold scheme*, we divide coflows into narrow and wide coflows according to their width, and adjust their priorities according to different types of thresholds. Note that, this approach considers both the coflow width and the sent bytes of every flow.

We have implemented MCS in a trace-driven simulator [16], and evaluate its performance with realistic Facebook workloads [17]. Our simulation results show that compared to the state-of-the-art information-agnostic scheme Aalo [10], MCS reduces the Coflow Completion Time (CCT) by up to 62% for $short\ \&\ narrow$ coflows, and achieves up to 17% lower CCT for $short\ \&\ wide$ coflows.

The main contributions of MCS are:

- We investigate that the coflow completion time could be further reduced by leveraging multiple coflow attributes.
- We propose MCS, a simple, effective, information-agnostic coflow scheduler to optimize the average CCT in data center networks.

II. Background and Related work

A. Attributes of Coflows

Unlike the traditional flow abstraction, a coflow consists of multiple parallel flows that have independent input and output. Thus, it has several attributes, and we will define some important coflow attributes as shown in Figure 1. We assume for a Coflow k denoted as $D^{(k)}$, it has a set of flows $\{d_{i,j}^{(k)}\}$, where $d_{i,j}^{(k)}$ denotes a flow from source i sends $d_{i,j}^{(k)}$ size data to destination j. Then, the $size$, $length$ and $width$ of a coflow k is defined as $d^{(k)} = \sum_{i,j} d_{i,j}^{(k)}$, $l^{(k)} = \max\{d_{i,j}^{(k)}\}$, and $w^{(k)} = |\{d_{i,j}^{(k)}\}|$, respectively. We can see that the $size$ of a coflow is the sum of all its flows' size, the $length$ of a coflow is the size of its largest flow, and the $width$ of a coflow is the number of parallel flows. For simplicity, we use $\widetilde{d_{i,j}^{(k)}}$, $\widetilde{d^{(k)}}$ and $\widetilde{l^{(k)}} = \max\{\widetilde{d_{i,j}^{(k)}}\}$ to represent the sent bytes of a flow, the sent bytes of a coflow and the length of a coflow in the information-agnostic scenario.

B. Related work

There is a large number of related work about coflow scheduling in data center networks. We only summarize some of them and mainly introduce information-agnostic coflow schedulers that are most related to our work.

Information-aware coflow scheduler Orchestra [6] and Varys [4] are both centralized coflow schedulers using heuristic algorithms to optimize the average CCT. Due to the scalability problem of the centralized scheduler, D-CAS [18] and OPTAS [9] propose distributed scheduling algorithms to improve the performance. Because the above heuristic algorithms are sub-optimal, RAPIER [7] and [8] use optimization theory to solve the coflow scheduling problem.

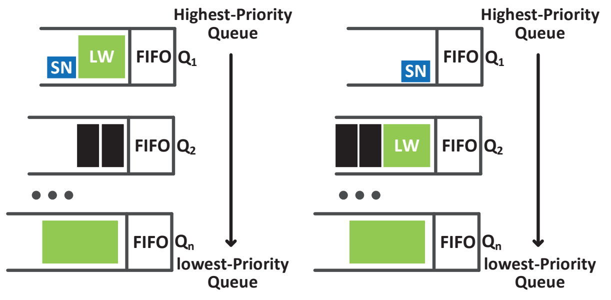
Information-agnostic coflow scheduler Aalo [10] is the first information-agnostic coflow scheduler and proposes Discretized Coflow-Aware Least-Attained Service (D-CLAS) to divides coflows into multiple priority queues. More specifically, a coflow within higher-priority Q_n is demoted to lower-priority Q_{n+1} when its sent bytes $\widetilde{d^{(k)}}$ crosses the predefined threshold θ_n. Then, coflows within each queue are scheduled in the First-In-First-Out (FIFO) order, and flows within each coflow use max-min fairness. Because Aalo needs to modify data-parallel applications to identify coflows, CODA [13] attempts automatically identify coflows without any application modifications and proposes error-tolerant coflow scheduling algorithm. In addition, [14] theoretically analyzes how to choose the thresholds of queues and proposes the Down-hill searching (DHS) algorithm.

III. Motivation

A. Impacts of HOL blocking

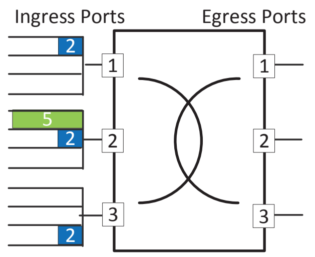
Because using sent bytes cannot differentiate newly-arrived small coflows and large coflows, existing information-agnostic coflow schedulers, such as Aalo [10], CODA [13], simply assign all the newly-arrived coflows with the highest priority. However, this policy may cause HOL blocking in the presence of large coflows.

To illustrate this, we consider a simple scenario in Figure 2, where a long and wide coflow (C_1) and a short and narrow coflow (C_2) sharing the same bottleneck link arrive at the same time. In Figure 2a, we assume that the scheduler assigns C_1 and C_2 with the highest priority and places them in the highest-priority queue Q_1 when they arrive. Then, we can see that C_2 is blocked by the larger coflow C_1 until the sent bytes of C_1 exceed the threshold of Q_1. Because the threshold of Q_1 is usually larger than 10MB [10], the CCT of C_2 will greatly increase. In Figure 2b, a well-designed scheduler classifies newly-arrived coflows based on their widths and estimated lengths, and then places large coflows into lower-priority queues and places small coflows into higher-priority queues. In this way, the HOL blocking can be avoided. Note that the above example is not a corner case. Because 52%



(a) With the highest priority (b) With different priorities

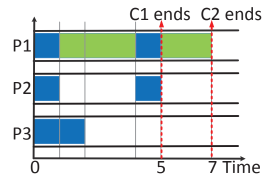
Fig. 2: Impact of HOL blocking. Long-wide coflow C_1 in green is scheduled before short-narrow coflow C_2 (blue)



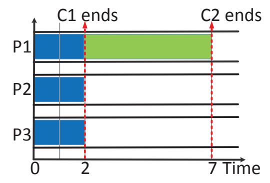
(a) Datacenter fabric

	C1	C2
Coflow Arrival Time	0	1

(b) Coflow arrival times



(c) Considering the coflow sent bytes



(d) Considering multiple coflow attributes

Fig. 3: Impact of only relying on sent bytes.

coflows are short and narrow while 17% coflows that create 99% traffic are wide and long [4].

Observation 1 *In the presence of large coflows, assigning newly-arrived coflows with different priorities can avoid HOL blocking.*

B. Impacts of only relying on the sent bytes

In previous information-agnostic coflow scheduling algorithms, a coflow's priority is gradually decreased if its sent bytes exceeds the predefined thresholds. However, the problem of this method is that it may increase the average CCT, especially when coflows have very different width.

Consider Figure 3 as an example. As previous work does [4], [10], [12], we abstract the data center network as a non-blocking switch. In Figure 3a, a large coflow C_1 and a small coflow C_2 are waiting for transmitting in the network. Figure 3b shows their arrival time.

Only relying on the sent bytes Figure 3c illustrates the scheduling result of Aalo. We assume there are two coflow queues (Q_1 and Q_2), and the threshold of Q_1 is 3 bytes. Therefore, C_1 will demote to Q_2 at $1s$ when it sends 3 bytes data. Then, C_2 within Q_1 starts transferring and its sent bytes will reach the threshold at $4s$. In this schedule, **the average CCT is $\frac{5+7}{2} = 6s$.**

Considering multiple coflow attributes Unlike Aalo, in Figure 3d, the well-designed scheduler integrally considers *the width, length, and size* of coflows. First, the scheduler classifies C_1 as the wide coflow and C_2 as the narrow coflow based on their widths. Then, the scheduler treats these two types of coflows differently. More specifically, for narrow coflows, the scheduler adjusts the priority based on the coflow size. For wide coflows, the scheduler changes the priority based on the coflow length. If the threshold is also 3 bytes, in this schedule, **the average CCT is $\frac{2+7}{2} = 4.5s$.**

The reason for the above problem is that the sent bytes of narrow coflows and wide coflows change in different ways. For narrow coflows, their sent bytes increases slowly, while for wide coflows, their sent bytes increases more rapidly since they contain a large number of flows and their data can be sent in parallel. Besides, we can see from Figure 4b, the size CDF of narrow coflows is very different from the size CDF of wide coflows. Therefore, it is too crude to adjust their priorities using the same threshold and method.

Observation 2 *Adjusting coflows' priorities is not only related to the coflow size but also related to other attributes.*

IV. Design Details

The goal of MCS is to design an information-agnostic coflow scheduler that leverages multiple coflow attributes. Like previous information-agnostic schedulers, we assume the length and size of a coflow cannot be known prior, and the width of each coflow can be obtained. Note that the second assumption is reasonable, because the width attribute can be directly obtained if the data-parallel applications implement the coflow schedulers' API [10] or the width attribute can be measured by clustering [13].

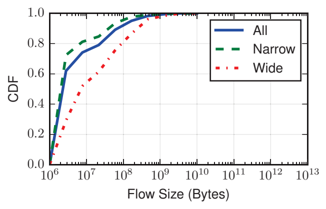
A. Shortest and Narrowest Coflows First

Observation 1 indicates that assigning newly-arrived coflows with different priorities is the key to reduce the average CCT of small coflows. Thus, we propose the Shortest and Narrowest Coflows First (SNCF) algorithm that the *short-narrow* coflows should be assigned higher-priority than *long-wide* coflows.

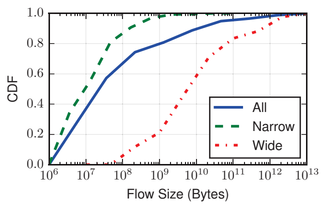
At the heart of this algorithm is assigning coflows the highest-priority when their $width$ is smaller than a threshold Δ. We use the Bayesian probability theory [19] to analyze this problem and find the optimal partition threshold Δ. We denote the cumulative density function of the coflow size, coflow width, coflow length as $F_{\mathbf{S}}(x), F_{\mathbf{W}}, F_{\mathbf{L}}(x)$, respectively. We assume a coflow is short if its longest flow is less than Δ_l and narrow if it involves at most Δ_w flows. Then, if Δ_l is given, our object is to find the smallest Δ that satisfies the following constraints:

$$\min \quad \Delta \tag{1}$$

$$\text{subject to} \quad \frac{F_{\mathbf{L},\mathbf{W}}(\Delta_l, \Delta)}{F_{\mathbf{W}}(\Delta)} \geq \alpha \tag{2}$$



(a) Coflow length distributions



(b) Coflow size distributions

Fig. 4: Splitting coflows into $narrow$ and $wide$ based on $\Delta = 50$.

where $F_{\mathbf{L},\mathbf{W}}()$ is the joint cumulative distribution of the coflow length and width. Equation 2 is the condition probability that a coflow is short on the given that a coflow's wide is smaller than Δ. If this probability is greater than α (e.g., 80%), we can set the Δ to the solution of Equation 1.

Actually, if we set Δ_l to 5MB as Varys [4] does, a wide range of Δ can satisfy the Equations 1 and 2. For example, for the Facebook workload [17], $\Delta = 50$ is a good solution that satisfies the subjection. To better visualize this, we split coflows into narrow and wide coflows based on $\Delta = 50$ and show their size and length distributions in Figure 4. As expected, we can see that the lengths of more than 80% narrow coflows are smaller than 5MB. In other words, if we know the wide of a coflow is small, its length is also high likely to be small.

B. Double-Thresholds

Through observation 2, we can find that the priorities of coflows are influenced by multiple coflow attributes and using one type of demotion thresholds is too crude to adjust the priority. Therefore, we present a double-thresholds scheme to adjust the priority.

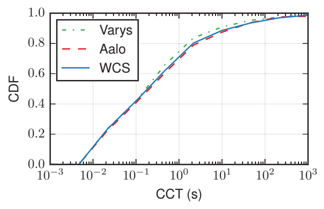
In the double-thresholds scheme, we adjust wide coflows' priorities based on their length $\widetilde{l_{i,j}^{(k)}}$ and adjust narrow coflows' priorities based on their sent bytes $\widetilde{d_{i,j}^{(k)}}$. Then, the demotion-thresholds-vector for wide coflows and narrow coflows are denoted as $\mathbf{\Theta}^w = [\theta_1^w, \theta_2^w, ..., \theta_n^w]$ and $\mathbf{\Theta}^n = [\theta_1^n, \theta_2^n, ..., \theta_n^n]$. The setting of each threshold in our algorithm can use the previous standards, such as uniformly-spaced scheme [4], exponentially-spaced scheme [4], DHS scheme [14]. For instance, we can use the uniformly-spaced scheme to choose $\mathbf{\Theta}^w$ while use the exponentially-spaced scheme to choose $\mathbf{\Theta}^n$. Although the settings of thresholds have impacts on the performance of MCS, exploring the optimal settings of thresholds is beyond the scope of this paper.

C. MCS Scheduler

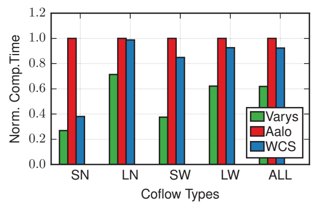
We extend the general framework of a non-clairvoyant coflow scheduler described in Aalo [10] and present MCS. MCS mainly has two stages to schedule coflows as shown in Algorithm 1.

Algorithm 1 The Main MCS Algorithm

```
1:  function NewCoflow(Queues $Q$, Coflow $k$)
2:      for $i \in 1...n-1$ do
3:          if $w^{(k)} \leq \Delta_i$ then
4:              put $k$ into $Q_i$
5:              Break
6:          end if
7:      end for
8:  end function
9:  function Reschedule(Queues $Q$)
10:     for $i \in 1...n-1$ do
11:         for Coflow $k \in Q_i$ do
12:             if $w^{(k)} \leq \Delta_w$ and $\widetilde{d^{(k)}} > \theta_i^n$ then
13:                 Remove $k$ from $Q_i$ and put $k$ into $Q_{i+1}$
14:             end if
15:             if $w^{(k)} > \Delta_w$ and $\widetilde{l^{(k)}} > \theta_i^w$ then
16:                 Remove from $Q_i$ and put to $Q_{i+1}$
17:             end if
18:         end for
19:     end for
20: end function
```



(a) CCT distributions



(b) The Average improvements in CCTs normalized to Aalo

Fig. 5: CCT for Varys, Aalo, and MCS

1) Assigning Initial Priority: The function NewCoflow sets the initial priority for each newly-arrived coflow based on their width. Note that, instead of directly using one Δ to prioritize coflows into two priority classes, we prioritize coflows into more than two priority classes by extending Δ. More specifically, the threshold of Q_i is $\Delta_i = \Delta \times E^{(i-1)}$.

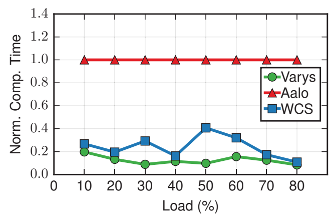
2) Adjusting Priority: When new coflows arrive or old coflows finish, we will use the function Reschedule to adjust the priorities of all the coflows. In our scheduler, we directly adopt the priority queues of Aalo [10], which uses three scheduling disciplines: 1) across queues, coflows from higher-priority queues are scheduled first, 2) within each queue, coflows are scheduled based on FIFO, 3) flows within each coflow, Max-Min principle is used.

V. Evaluation

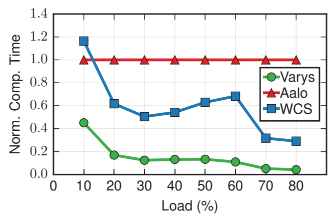
A. Simulation Settings

Simulator: We implement MCS in a trace-driven flow-level simulator used by Aalo [16]. This simulator performs a detailed replay of the coflow traces.

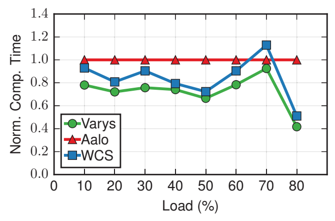
Workload: We use a realistic workload from Facebook



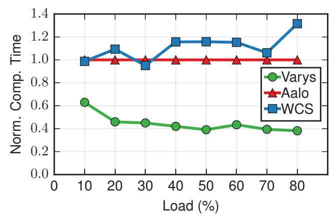
(a) Short and Narrow



(b) Long and Narrow

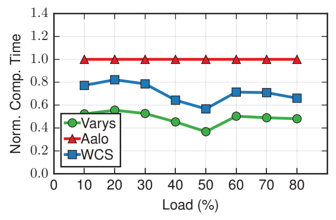


(c) Short and Wide

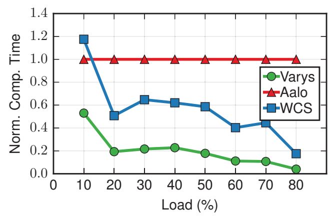


(d) Long and Wide

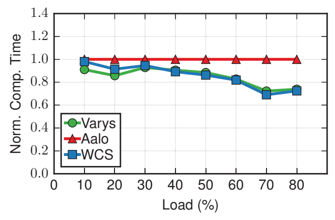
Fig. 6: CCT when batch interval is 10 s.



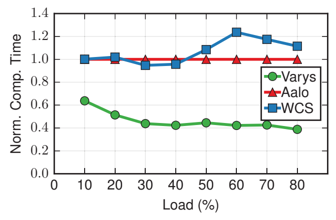
(a) Short and Narrow



(b) Long and Narrow



(c) Short and Wide



(d) Long and Wide

Fig. 7: CCT when batch interval is 1 s.

Coflow Bin	SN	LN	SW	LW
% of Coflows	52%	16%	15%	17%
% of Bytes	0.01%	0.67%	0.22%	99.10%

TABLE I: Coflows binned by their length (Short and Long) and their width (Narrow and Wide).

production cluster [17].The coflow size and length distribution of the workload are shown in Figure 4. Coflows are classified into four types based on their width and length as shown in Table I. Basically, a coflow is *short* if its longest flow is less than 5 MB and *narrow* if it has at most 50 flows [4]. Note that, similar to the settings of Varys, coflows arrive in batch and the default batch interval is 10s.

Metrics: We measure the coflow completion time (CCT), and use the average CCT normalized to Aalo as the main performance metric.

Parameters of MCS: We use exponentially-spaced scheme [10] to set the thresholds: $\Delta_i = 10^i$, $\theta_i^n = 10^i$ MB and $\theta_i^w = 2^{i-1}$ MB.

B. Results under Normal Workloads

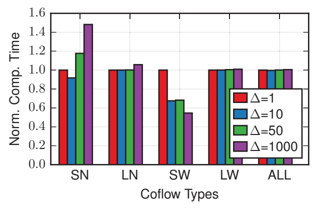
We firstly investigate the performance of MCS under the Facebook workload. Figure 5b shows the performance of different scheduling algorithms in terms of the average CCT normalized to Aalo. It implies that MCS performs as well as information-aware coflow scheduler Varys for *short* and *narrow* coflows, and significantly outperforms information-agnostic coflow scheduler Aalo. For example, MCS reduces the average CCT for $short\&narrow$ coflows by up to $\sim 62\%$ and reduces the average CCT for $short\ \&\ wide$ coflows by up to $\sim 17\%$. In addition, it is not a surprise that Varys performs the best since it knows coflow information prior. Figure 5a shows the CDFs of CCT under different scheduling algorithms. We can see that MCS is between Varys and Aalo.

C. Impact of Load

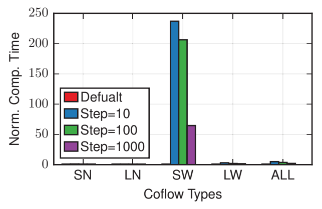
To study the impact of network load, we change the coflow arrival time in the Facebook trace according to the desired load. Basically, coflows arrive according to a Poisson process, and the arrival rate is set to $\lambda = \frac{load \times NetBandwidth}{MeanCoflowSize}$. The simulation results by varying the network load from 10% to 80% are shown in Figure 6.

Figure 6a shows that MCS achieves very good performance for $short\&narrow$ coflows. Compared to Aalo, MCS reduces the average CCT for $short\&narrow$ coflows by up to $\sim 90\%$. This is due to the fact that the SNCF mechanism of MCS helps identify small newly-arrived coflows and only gives them the highest-priority to avoid blocking while Aalo gives all newly-arrived coflows the highest priority. In addition, MCS also achieves good performance for $long\,\&\,narrow$ coflows. With the increase of the load, the performance gap between MCS and Aalo becomes large, and MCS achieves 70% lower CCT at 80% load.

Figure 6c shows that MCS has very similar performance with Varys for $short\&wide$ coflows. Compared to Aalo, MCS reduces the average CCT for $short\ \&\ wide$ coflows by up to 50%. Note that the performance gap between MCS and Varys is small. On the contrary, MCS achieves worst performance for $long\ \&\ wide$ coflows at high loads. This problem is mainly caused by the large batch interval. When large coflows and small coflows arrive in batch, because MCS gives higher-priority to small coflows, the CCT of large coflows will certainly increase. As shown in Figure 7d, if we use small batch interval, the performance gap is very small at low loads.

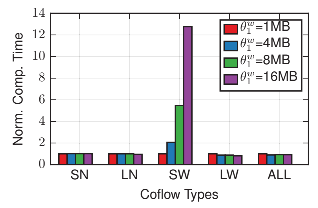


(a) $\Delta_i = \Delta * 10^{(i-1)}$

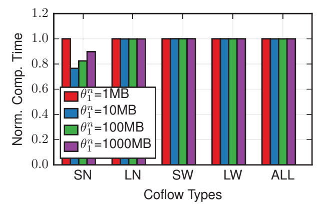


(b) $\Delta_i = step \times i$

Fig. 8: Impact of Δ



(a) $\theta_i^w = \theta_1 \times 2^{i-1}$



(b) $\theta_i^n = \theta_1 \times 10^{i-1}$

Fig. 9: Impact of $\mathbf{\Theta}^w$ and $\mathbf{\Theta}^n$.

D. Impact of Batch arrival

The above subsection has studied the performance of MCS under different load. In this experiment, to study the impact of batch arrival, we change the batch interval to 1 s and repeat the previous simulation.

Figure 7 shows the simulation results. Compared to Figure 6, we observe on obvious degradation in the performance gap for $short$ & $narrow$ coflows. For example, the performance gap between MCS and Aalo decreases from 90% to 30% at 80% load. This is because when the batch interval is small, the number of newly-arrived coflows in a batch becomes small, making small newly-arrived coflows are less likely blocked by large coflows. For the other types coflows, the performance of MCS and other coflow scheduling algorithms are similar to the result shown in Figure 6.

E. Sensitivity to Parameters

We now examine MCS's sensitivity to parameters. Recall that MCS mainly has three parameters to configure: Δ, $\mathbf{\Theta}^w$ and $\mathbf{\Theta}^n$.

The first parameter Δ is used to assign the initial coflow priority. As we increase Δ, more coflows will be classified as small coflows and given higher-priority. In Figure 8a, we can find that when Δ is between 10 and 50, MCS achieves a good performance, while too large and small Δ decrease the performance. Then, instead of increasing Δ_i exponentially, we use a uniformly-space scheme to set Δ_i. In the uniformly-space scheme, Δ_i is set to $step \times i$. The simulation results in Figure 8b show that uniformly-space scheme achieves very bad performance compared to the exponentially-space scheme. This is because that when the step is small, a large number of coflows will be assigned the lowest-priority.

The other two parameters are used to adjust the priorities of coflows during their transmission. First, we focus on how $\mathbf{\Theta}^w$ impact on MCS's performance. As we increase θ_1^w from 1MB to 16MB, more $wide$ coflows will be scheduled in the same queue. Figure 9a shows that increasing θ_i will greatly decrease the performance of $short\,\&\,wide$ coflows. Then, We change θ_1^n from 1MB to 1000 MB. As we increase θ_i^n, more $short$ coflows will be scheduled in the same queue. As shown in Figure 9b, $\mathbf{\Theta}^n$ has fewer impacts on the performance of $short\,\&\,narrow$ coflows since they have the highest-priority.

VI. Conclusion

This article showed that the design space for information-agnostic scheduling should not be limited to the sent-bytes-based rate allocation. Based on this, we have presented MCS that leveraging multiple coflow attributes to further reduce the completion time of coflows. Numerical simulation results demonstrated that MCS could greatly reduce the completion time of small coflows and outperformed the previous information-agnostic coflow scheduling algorithm Aalo.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," in *SIGOPS*. ACM, 2007, pp. 59–72.
[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-memory Cluster Computing," in *NSDI*, 2012, pp. 2–2.
[4] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *SIGCOMM*. ACM, 2014, pp. 443–454.
[5] M. Chowdhury and I. Stoica, "Coflow: a Networking Abstraction for Cluster Applications," in *Hotnets*. ACM, 2012, pp. 31–36.
[6] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing Data Transfers in Computer Clusters with Orchestra," in *SIGCOMM*. ACM, 2011, pp. 98–109.
[7] Y. Zhao, K. Chen, W. Bai, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *INFOCOM*, 2015.
[8] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing Coflow Completion Times with Utility Max-Min Fairness," in *INFOCOM*. IEEE, 2016.
[9] Z. Li, Y. Zhange, D. Li, K. Chen, and Y. Peng, "OPTAS: Decentralized Flow Monitoring and Scheduling for Tiny Tasks," in *INFOCOM*. IEEE, 2016.
[10] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *SIGCOMM*. ACM, 2015, pp. 393–406.
[11] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized Task-aware Scheduling for Data Center Networks," in *SIGCOMM*. ACM, 2014, pp. 431–442.
[12] Z. Huang, B. Balasubramanian, M. Wang, T. Lan, M. Chiang, and D. H. Tsang, "Need for Speed: CORA Scheduler for Optimizing Completion-Times in the Cloud," in *INFOCOM*. IEEE, 2015, pp. 891–899.
[13] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark," in *SIGCOMM*. ACM, 2016, pp. 160–173.
[14] Y. Gao, H. Yu, S. Luo, and S. Yu, "Information-agnostic Coflow Scheduling with Optimal Demotion Thresholds," in *ICC*. IEEE, 2016, pp. 1–6.
[15] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic Flow Scheduling for Commodity Data Centers," in *NSDI*, 2015, pp. 455–468.
[16] "CoflowSim," https://github.com/coflow/coflowsim.
[17] "Coflow-Benchmark," https://github.com/coflow/coflow-benchmark.
[18] S. Luo, H. Yu, Y. Zhao, B. Wu, S. Wang *et al.*, "Minimizing average coflow completion time with decentralized scheduling," in *ICC*. IEEE, 2015, pp. 307–312.
[19] J. M. Bernardo and A. F. Smith, "Bayesian theory," 2001.