Coflow Scheduling of Multi-stage Jobs with Isolation Guarantee

Zifan Liu, Haipeng Dai and Wanchun Dou
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, China
zifanliu@smail.nju.edu.cn, haipengdai,douwh@nju.edu.cn

TABLE I
KEY TERMS AND DESCRIPTIONS

Terms	Description
M	The number of total jobs.
N	The number of total coflows.
K	The number of machines.
$F_i = \langle f_i^1, \dots, f_i^{2K} \rangle$	Demand vector of coflow-i.
$d_i = \langle d_i^1, \dots, d_i^{2K} \rangle$	Correlation vector of coflow-i.
$a_i = \langle a_i^1, \dots, a_i^{2K} \rangle$	Bandwidth allocation of coflow-i.
$\overline{f_i} = \max_k f_i^k$	Bottleneck demand of coflow-i.
P_i	Progress of coflow-i.
Γ_m	Progress of job-m.

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—keyword, keyword, keyword**

I. INTRODUCTION

This document is a model and instructions for LaTeX. Please observe the conference page limits.

II. MODEL AND OBJECTIVE

In this section, we firstly delineate the model of datacenter networks and coflow and then discuss two objectives. To simplify the discussion, key terms used in our model are summarized in Table 1.

A. Model

Given the full bisection bandwidth, which has been well developed in modern datacenter [1], we treat the datacenter network as a big no-blocking switch connecting K machines. Each machine has one ingress port and one egress port, thus the whole fabric has $2K$ ports. In this simplified model, the edges are the only place for congestion. Hence we focus sorely on bandwidth of each port. In our analysis, all links are assumed of equal capacity normalized to one.

The coflow abstraction presents the communication demand within two stages of parallel computing model. A coflow is composed of a collection of flows across a group of machines sharing a common performance requirement. The completion time of the latest flow defines the completion time of this coflow. In many data-parallel frameworks like MapReduce/Hadoop, the coflow properties, such as source, destination, amount of data transferred of each flow, are known as a priori [2]–[4].

Specifically, the coflow *demand vector* $F_i = \langle f_i^1, \dots, f_i^{2K} \rangle$ captures the data demand of coflow-i, where f_i^k denotes the amount of data transferred on port k. Additionally, f_i^{hl} denotes a flow transferring data from port h to port l. Among all flows in coflow-i, we name the port with largest traffic bottleneck port. Let the data demand on this port be the *bottleneck demand*, defined as $\overline{f_i} = \max_k f_i^k$. To simplify our analysis, the *correlation vector* $d_i = \langle d_i^1, \dots, d_i^{2K} \rangle$ is engaged to describe the demand correlation across ports, where d_i^k is the normalized data demand on port k by the bottleneck demand, i.e., $d_i^k = f_i^k / \overline{f_i}$. This vector indicates that for every byte coflow-i sends on bottleneck port, at least d_i^k bytes should be transferred on port k.

Coflows have elastic bandwidth demand on multiple ports, comparing with individual flows. Given the bandwidth allocation vector $a_i = \langle a_i^1, \dots, a_i^{2K} \rangle$ for each coflow-i calculated by coflow scheduler given the demand vectors, the coflow progress is restricted by the worst-case port. Formally, *progress* of coflow-i is measured as the minimum demand-normalized rate allocation across ports, i.e.,

$$P_i = \min_{i:d_i^k>0} \frac{a_i^k}{d_i^k}. \tag{1}$$

Intuitively, progress of coflow-i means the transmission satisfaction ratio on the lowest port, which determines the CCT of coflow-i.

Assume a multi-stage job-m is a collection of N coflows, i.e., $J_m = \{c_{m,1}, \dots, c_{m,N}\}$. Given the bottleneck demand and progress of each coflow, i.e., $\{P_{m,1}, \dots, P_{m,N}\}$, the progress of job-$m$ can be computed as

$$\Gamma_m = \frac{\sum_{n=1}^{N} \overline{f_{m,n}} P_{m,n}}{\sum_{n=1}^{N} f_{m,n}}. \tag{2}$$

Like above, progress of job-m indicates the collectivity transmission satisfaction ratios of all coflows belonging to it, which has significant effect on the JCT of job-m.

B. Objective

In common consensus [5], a coflow scheduler focuses primarily on two objectives, average CCT and isolation guarantee. Under the multi-stage coflow scheduling problem, we should concern the average JCT instead.

1) *Average JCT*: To speed up data-parallel application completion time, as many jobs as possible should be finished in their fastest possible ways. Therefore minimizing the average JCT is settled as a critical objective for an efficient coflow scheduler.
2) *Isolation Guarantee*: In a shared datacenter network, all tenants expect *performance isolation guarantees*. Existing work has define such guarantee as the *minimum progress* across coflows [6], i.e., $\max_i P_i$. For multi-stage jobs, we define the isolation guarantee as the minimum progress across jobs, i.e., $\max_m \Gamma_m$. To optimize the isolation guarantee, a coflow scheduler should look for an allocation to maximize the minimum progress.

However, the coflow isolation guarantee concludes the real-time progress of active coflows, while the multi-stage job isolation guarantee needs the monolithic view of all coflows it consisting of. Thus we introduce the job *long-term isolation guarantee* to our model.

Definition 1 (Long-term Isolation Guarantee): Consider a multi-stage job-i, let T_i be its JCT by coflow scheduler S. T_i^* is its JCT by a fair scheduler which enforce a minimum instantaneous progress of all active coflows. We call the scheduler S provides the job long-term isolation guarantee if all jobs complete no later than $T_i^* + D$, where D is a constant delay, i.e.,

$$T_i \leq T_i^* + D \tag{3}$$

In this paper, our objective is to gain the best of both targets in a long run, which is formalized as

$$\begin{aligned} \text{minimize} \quad & \sum_i T_i \\ \text{s.t.} \quad & T_i \leq T_i^* + D, \text{ for all job-}i \end{aligned} \tag{4}$$

III. Algorithm and Analysis

In this section, we designed a coflow scheduling algorithm of multi-stage jobs with isolation guarantee. We first elaborate how to sort coflows based on their priorities. Second we promote our bandwidth allocation algorithm.

A. Coflow Sorting

Before we allocate bandwidth to coflows, the priorities of each coflow should be determined. Consider all coflows are scheduled by the fair scheduler DRF, which enforces the equal progress across coflows [6], [7]. On account of the completion time of each coflow under such scheduling, we can obtain a priority order for all coflows. If a coflow completes faster in DRF scheduling, it has a higher priority than the lowers.

However, directly performing DRF is not suitable in the context of multi-stage jobs, because some coflows that have dependencies are not released at the start. At first, we can only calculate the completion times of coflows F' with no dependencies, which are released at time 0. Then, the minimum progress of these coflows $P^* = \min_i P'_i$ is produced. It is noticed that in popular parallel frameworks, the number of coflows in different stages stays similar [8], [9]. We assume that P^* keeps across stages, then the completion time of coflow $F_i \notin F'$ is estimated as $t_i = P^* \overline{f_i}$.

Besides, the completion time of single coflow is not necessarily the priority of the job containing it. In multi-stage jobs, coflow pipelines are developed using directed acyclic graph(DAG) pattern to show the dependencies between coflows. Setting the weights of coflows in DAG as their completion times, the completion time of job-m is the maximum total weight along DAG routes of J_m. Like previous, if a job completes faster, it has a higher priority. Within a job, the priority order of coflows is the topological order, along with giving the faster coflows higher priority. Therefore we get a prioritized queue of all coflows $\mathbf{F} = (F_1, \ldots, F_N)$. The entire procedure is summarized in Algorithm 1.

Algorithm 1 Coflow Sorting Algorithm

```
Input: Data demand set of all coflows $F$ and the job set $J$.
Output: An ordered queue of all coflows $\mathbf{F}$.
 1: $F' \leftarrow$ coflows have no dependencies in $F$.
 2: $P^* \leftarrow$ the minimum progress of $F'$ under DRF.
 3: for $i = 1 \rightarrow |F|$ do
 4:     if $F_i \in F'$ then
 5:         $t_i \leftarrow$ the completion time of $F_i$ under DRF.
 6:     else
 7:         $t_i \leftarrow P^* \overline{f_i}$.
 8: $T_m \leftarrow$ the maximum total time along DAG route of $J_m$.
 9: Sort $J$ in the ascending order of $T$.
10: Initialize $F^*$ as an empty queue.
11: for $m = 1 \rightarrow |J|$ do
12:     Sort coflows $\overline{F_m} = \{F_i | F_i \in J_m\}$ in topological order
        of DAG.
13:     $\mathbf{F} \leftarrow \mathbf{F} + \overline{F_m}$
```

B. Bandwidth Allocation

After obtaining the prioritized queue of coflows, we can allocated bandwidth to the active coflows one by one. Specifically, we introduce the *super-coflow* conception which was proposed in [10] for our solution.

Super-coflow. Given a queue of coflows $\mathbf{F}$, the super-coflow S_i is defined as the sequential aggregation of the first i coflows $F_1, \ldots, F_i$. Formally, the demand vector $D_i = \langle D_i^1, \ldots, D_i^{2K} \rangle$ of S_i is the accumulation of the first i coflows, i.e., $D_i = \sum_{j=1}^{i} d_j$. Super-coflow S_1 trivially degrades into coflow F_1 as a special case.

In previous works, coflows are mostly scheduled individually. We take the minimum-allocation-for-desired-duration (MADD) algorithm as an example [2]. Under MADD, the least amount of bandwidth was allocated to a coflow to obtain the maximum possible progress. Let R_k be the remaining bandwidth on port k and $d_i = \langle d_i^1, \ldots, d_i^{2K} \rangle$ be the correlation vector of coflow-i. Thus the maximum possible progress P' of coflow-i can be calculated as

$$P' = \min_{1 \leq k \leq 2K} \frac{R_k}{d_i^k}. \tag{5}$$

To acquiring the maximum possible progress, the bandwidth allocation of coflow-i on port k is at least $P'd_i^k$. Under MADD, the allocation is exactly $P'd_i^k$. However, MADD may lead to the priority inversion problem. If a coflow is not able to get any progress, then it gets no bandwidth, even if allocating it some bandwidth will accelerate its completion.

On the contrary, our super-coflow conception eliminates the priority inversion situation as much as possible. When computing the bandwidth allocatio, coflow F_i is justified if it contributes to super-coflow S_i. As long as F_i affects the progress of S_i, a bandwidth allocation to F_i is always guaranteed, even when it gains no progress alone, which easily triggers a priority inversion under MADD.

Assume the active coflow priority queue is $F' = (F'_1, \ldots, F'_{N'})$ and the super-coflow queue is $S = (S_1, \ldots, S_{N'})$. Our bandwidth allocation algorithm runs in turns. In the i-th turn, we will allocate the least bandwidth to F_i, while the allocations of earlier turns stay unchanged, in order to achieve the minimum possible completion time of S_i. To easily describe our algorithm, we next focus on the bandwidth allocation of F_i.

To achieve the minimum possible completion time of S_i, we first calculate the bottleneck data demand of S_i, i.e.,

$$\overline{D_i} = \max_{1 \le k \le 2K} D_i^k. \tag{6}$$

Let D_i^{hl} be the data amount transferred from port h to port l in S_i. Then at least $D_i^{hl}/\overline{D_i}$ bandwidth should be allocated for the flows between port h and port l. It is noted that the first $i-1$ coflows have received some bandwidth in the earlier turns. We denote the bandwidth received by flow f_k^{hl} in coflow-k as u_k^{hl} for $k < i$, thus we can distribute at most $(D_i^{hl}/\overline{D_i} - \sum_{k=1}^{i-1} u_k^{hl})^+$ bandwidth to f_i^{hl}, where $(x)^+ = \max(0, x)$. Given the remaining bandwidth R_h on port h and R_l on port l, which limit the actual bandwidth available for f_i^{hl}, we finally get u_i^{hl} as

$$u_i^{hl} = \min\left[\left(D_i^{hl}/\overline{D_i} - \sum_{k=1}^{i-1} u_k^{hl}\right)^+, R_h, R_l\right]. \tag{7}$$

After then, the remaining bandwidth on these two ports are updated. When all turns have finished, unused bandwidth is distributed to coflows. For each ingress port h, remaining bandwidth R_h are distributed to corresponding flows according to ratio of their current received bandwidth u_i^{hl}, restricted by the corresponding egress port l. We summarize the whole procedure as $\mathsf{SuperflowAllocation}(F)$ in Algorithm 2.

When an old coflow is finished or a new coflow is released, the active coflow set F' is changed and the bandwidth allocations will be rescheduled. All the active coflows are sorted as the order we get by Algorithm 1 to maintain the priorities of all jobs through our scheduling. The main procedure is summarized as $\mathsf{BandwidthAllocation}(F)$ in Algorithm 2.

C. Long-term Isolation Guarantee

Our algorithm provides the job long-term isolation guarantee of Definition 1. For each job-i, the completion time T_i is guaranteed not to exceed a constant time over its completion time in DRF. We formalize this theorem as following:

Algorithm 2 Bandwidth Allocation Algorithm

```
1: procedure SUPERFLOWALLOCATION(Coflows $F$)
2:     Initialize unused bandwidth $R_k \leftarrow 1$ on port-$k$
3:     for $i = 1 \rightarrow |F|$ do
4:         Assemble demands $D_i = \sum_{k=1}^{i} d_k$.
5:         $\overline{D_i} \leftarrow \max_k D_i^k$
6:         for all flow $f_i^{hl} \in F_i$ do
7:             $u_i^{hl} \leftarrow \min[(D_i^{hl}/\overline{D_i} - \sum_{k=1}^{i-1} u_k^{hl})^+, R_h, R_l]$.
8:             $R_h \leftarrow R_h - u_i^{hl}$.
9:             $R_l \leftarrow R_l - u_i^{hl}$.
10:    Allocate remaining bandwidth to coflows in the order of $F$ by FIFO.
11: procedure BANDWIDTHALLOCATION(Coflows $F$)
12:    $\mathbf{F} \leftarrow$ sorted $F$ by Algorithm 1.
13:    Initialize active coflows $F' \leftarrow \emptyset$.
14:    while True do
15:        $F' \leftarrow$ released coflows in the order of $\mathbf{F}$.
16:        if $F'$ is changed then
17:            Update remaining transfer demand of $F'$.
18:            SuperflowAllocation($F'$).
19:        if $F' == \emptyset$ then
20:            break.
```

Theorem 1 (Long-term Isolation Guarantee): Assume that jobs J are released at time 0. For all job $J_i \in J$, let T_i be the JCT of J_i in Algorithm 2, and let T_i^* be the completion time of J_i in DRF. The completion time delay is bounded as

$$T_i - T_i^* \le . \tag{8}$$

ACKNOWLEDGMENT

REFERENCES

[1] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 183–197.

[2] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, 2014, pp. 443–454.

[3] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 393–406.

[4] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *INFOCOM 2018-IEEE Conference on Computer Communications, IEEE*, 2018.

[5] M. Chowdhury and I. Stoica, "Coflow:a networking abstraction for cluster applications," in *ACM Workshop on Hot Topics in Networks*, 2012, pp. 31–36.

[6] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "HUG: Multi-resource fairness for correlated and elastic demands," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, Mar. 2016, pp. 407–424.

[7] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Nsdi*, vol. 11, no. 2011, 2011, pp. 24–24.

[8] 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 *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2012, pp. 2–2.

[9] S. Wang, J. Zhang, T. Huang, J. Liu, T. Pan, and Y. Liu, "A survey of coflow scheduling schemes for data center networks," *IEEE Communications Magazine*, 2018.

[10] B. Li, L. Wang, and W. Wang, "Utopia: Near-optimal coflow scheduling with isolation guarantee," in *INFOCOM 2018-IEEE Conference on Computer Communications, IEEE*, 2018.