A Partial-decentralized Coflow Scheduling Scheme in Data Center Networks

Shuli Zhang†, Yan Zhang†*, Ding Tang†, Zhen Xu†, Jingguo Ge†, Zhijun Zhao†
†State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, China
*Corresponding author, email: zhangyan80@iie.ac.cn

***Abstract*—In this paper, we propose CGM-PS, a *partial-decentralized*, *un-starving*, *work-conservative*, and *preemptive* coflow scheduling scheme to shorten the Coflow Completion Time (CCT) for TCP flows in data center networks (DCNs). In CGM-PS, we propose both inter- and intra- coflow scheduling policies. In inter-coflow scheduling, we present P-SEBF, which adopts a connected-graph model based novel concept *Partialcoflow*, to achieve approximate SEBF scheduling in a partial-decentralized manner. In intra-coflow scheduling, we present FP-MDFS to give flow-level priorities and appropriate rates to TCP flows for finishing the coflows as quick as possible without wasting network capacities in a decentralized manner. Trace-based simulation results show that, among existing coflow scheduling schemes, CGM-PS can achieve the minimal CCTs both on average and in the 90^{th} percentile. In brief, CGM-PS only brings about similar scheduling overhead with the decentralized schemes, while it can achieve the CCT performance even better than the near optimal centralized scheme.**

I. Introduction

In recent years, data center networks (DCNs) host a wide range of Internet applications and services, such as online queries, recommendations and social network information, etc. These applications and services usually work based on various distributed computing and storage frameworks (eg., MapReduce [1], Spark [2] and HDFS [3]). In these frameworks, a set of parallel data transfer procedures, which can be in the form of *shuffle*, *aggregation*, and *broadcast* [4], play important roles and can affect the performance of the frameworks significantly. A parallel data transfer procedure usually involves multiple parallel TCP flows. A procedure can complete its task only when all involved TCP flows finish data transfers. All TCP flows involved in one parallel data transfer procedure are abstracted as a *Coflow* [4]. Recent studies point out that shortening the *Coflow Completion Time* (CCT) [5–7] is vital to improve the quality of Internet applications and services.

For shortening CCT, we propose a novel coflow scheduling scheme named CGM-PS (Connected Graph Model based Partial-decentralized Scheduling), which is *partial-decentralized*, *un-starving*, *work-conservative*, and *preemptive*. To reduce CCT, our CGM-PS scheme provides not only inter-coflow but also intra-coflow scheduling. For inter-coflow scheduling, we propose a new policy named as *Partialcoflow based Smallest Effective Bottleneck First* (P-SEBF). In P-SEBF, we use a novel concept *Partialcoflow* to achieve approximate *Smallest Effective Bottleneck First* (SEBF) scheduling. For intra-coflow scheduling, we propose a new scheduling policy named *Flow Priority based Meeting Demands otherwise Fairness Sharing* (FP-MDFS). This policy gives flow-level priorities to TCP flows and allocates appropriate rates to TCP flows to finish the coflows as quick as possible without wasting network capacities. Both inter- and intra- coflow scheduling are automatically collaborated by coflows' master, senders, receivers and intermediate switches in a partial-decentralized manner. Cluster-trace based simulation results indicate that, compared with existing coflow scheduling schemes, CGM-PS can achieve the minimal CCTs both on average and in the 90^{th} percentile. Specifically, compared with the decentralized schemes, CGM-PS brings only a little higher scheduling overhead but can reduce the CCT to a fraction. And compared with the centralized scheme, CGM-PS can achieve even shorter CCT while bringing much lower scheduling overhead.

II. Design of the CGM-PS Scheme

A. Definitions and Framework

Undirected-Graph: Since a typical TCP connection consists of forward data transmission and backward ACK feedback transmission, a TCP flow can be viewed as an undirected edge. Given this situation, *a coflow and its related servers can be abstracted into an undirected-graph* $G = \langle V, E \rangle$, where all senders and receivers compromise the vertex set V and all involved TCP flows make up the edge set E.

Connected-Graph: According to the connectivity of a graph, an undirected-graph can be split into several connected subparts. *Each connected part is a connected-graph.*

Partialcoflow: *A partialcoflow is a collection of all TCP flows which can be abstracted into the edge set of a connected-graph.*

Here we outline the framework of CGM-PS. Our CGM-PS scheme is divided into two parts: inter- and intra- coflow scheduling. Inter-coflow scheduling refers to arrange coflow-level priorities to different coflows, while intra-coflow scheduling is to assign flow-level priorities and rates to TCP flows within each coflow.

B. Inter-coflow Scheduling Policy

The goal of P-SEBF is to approximatively achieve SEBF [5] in a partial-decentralized manner. Table I lists some key notations and their meanings used in P-SEBF. As superscript M or S only refers to the variable is maintained by the coflow-master or the server, we don't mean to present the notations

978-1-4673-6770-7/15/$31.00 ©2015 IEEE

TABLE I
KEY NOTATIONS IN P-SEBF

Not.	Description
L_i^S	Id of the *i*-th server in the same connected-graph with the server
D_i^S	Remaining data volume of the *i*-th server in the same connected-graph with the server
L^S	Id of the server that possess the max remaining volume of TCP flows
D^S	Max remaining volume of TCP flows among all servers
$g_{i,j}^S$	Flag which indicates whether the *j*-th flow has spread the latest remaining data volume on the *i*-th server to its receiver
P	Flag which indicates whether the coflow has been served by the network
I	Number of servers in the same connected-graph with the sender
J	Number of TCP flows on this sender

of variables with superscript M. Notice that, all notations are for a coflow.

1) Coflow-Master Operations: When a coflow arrives at the network, the corresponding coflow-master informs each sender and receiver the initial data volume of TCP flows on the servers in the same connected-graph. This distributing information is formed as (L_i^M, D_i^M) (i=1, 2, ..., I). Besides, all senders will be informed the coflow's initial bottleneck (L^M, D^M). The coflow-master piggybacks the distributing information on the header of the packets which are called the *advertising packets* and then sends out the distributing packets to all involved senders and receivers.

2) Sender Operations: Each sender involved in a coflow maintains a set of variables for this coflow, including I, J, (L_i^S, D_i^S) (i=1, 2, ..., I), (L^S, D^S), and $[g_{i,j}^S]$ (i=1, 2, ..., I; j=1, 2, ..., J). Each sender will send out a *scheduling packet* for each TCP flow every RTT. If the sending rate of the flow is positive, the scheduling packet is a data packet piggybacked by a *scheduling header*; otherwise, the scheduling packet is a packet which has no data but possesses a scheduling header. Similarly, a *scheduling ACK* is an ACK which is piggybacked by a scheduling header.

The sender works in the following steps.

- Step 1. When a new coflow arrives at the network, update its initial information:
  - 1.1. Upon receiving the informed information (L_i^M, D_i^M) (i=1, 2, ..., I) and (L^M, D^M) from the coflow-master, transform these values to local information (L_i^S, D_i^S) (i=1, 2, ..., I) and (L^S, D^S). Then update I and J.
  - 1.2. Set the matrix $[g_{i,j}^S]$ (i=1, 2, ..., I; j=1, 2, ..., J) to all 0, and set P to 0.
- Step 2. Every RTT, send out a scheduling packet to its receiver for each TCP flow:
  - 2.1. Get the coflow id C, local remaining volume D, flow id $j^{'}$, and server id $i^{'}$.
  - 2.2. if D is less than $D_{i'}^S$, update $D_{i'}^S$ and set $[g_{i',j'}^S]$ to 1.
  - 2.3. For $i \in 1, 2, \ldots, I$, if $[g_{i,j'}^S]$ equals to 1, add (L_i^S, D_i^S) to a scheduling header and then set $[g_{i,j'}^S]$ to 0.
  - 2.4. If the scheduling header is not empty, send out a scheduling packet to the flow's receiver.
- Step 3. On receiving a scheduling ACK, update the local information of the corresponding coflow:
  - 3.1. Get the coflow id C.
  - 3.2. Put all feedback info on (L_i^R, D_i^R) (the superscript R refers that this variable is maintained by receivers) into a collection T.
  - 3.3. For each pair of (L_i^R, D_i^R) in T, if D_i^R is less than D_i^S, set D_i^S to D_i^R and $[g_{i,j}^S]$ (j = 1, 2, ..., J) to 1.
- Step 4. Every δ, compute the coflow-level priorities for all coflows on this sender:
  - 4.1. Remove finished coflows from the local set of coflows $\mathbb{C}$.
  - 4.2. For each coflow C in $\mathbb{C}$, update the value of P based on whether it has been served till now.
  - 4.3. For each coflow C, if P equals to 0, keep (L^S, D^S) unchanged and set its coflow-level priority to D^S; if P equals to 1, set D^S to the max value of D_i^S (i=1, 2,..., I), L^S to the corresponding L_i^S, and set its coflow-level priority to D^S. This step plays the key role of P-SEBF.
  - 4.4. Use equation (1) to adjust the coflow-level priority for C.
- Step 5. Give all TCP flows coflow-level priorities.

As least-size scheduling will result in starving for some coflows whose sizes are much greater than others. So, to alleviate this problem, in Step 4.4 we adjust P-SEBF to a function of time to re-compute a coflow's priority:

$$priority = D^S \times \max(\phi, (1 - \frac{pass\_time}{N^* \times \varphi})) \quad (1)$$

where φ is the coflow's average arrival interval, N^* is an empirical parameter which reflects how many other ones this coflow should wait for. ϕ is a scalable threshold.

3) Receiver Operations: The receiver operations are similar to the sender operations. The core idea is to help the sender to gather information in a decentralized manner.

C. Intra-coflow Scheduling Policy

1) Sender Operations: In FP-MDFS, each sender computes a flow-level priority and a desired rate for each TCP flow on this sender. The flow-level priority is designed as a two-tuple (P_inter, P_intra), where P_inter is the coflow-level priority, and P_intra is a flag which indicates whether the transmission path of the flow is on its coflow's bottleneck link.

The desired completion time of a coflow can be got as follows:

$$desired\_CCT = \frac{D^S}{R} \quad (2)$$

where R denotes the NIC rate of a server.

Based on this, the desired rate of a TCP flow can be computed as follows:

$$desired\_rate = \min\left(\frac{d}{desired\_CCT}, \frac{DataInBuffer}{RTT}\right) \quad (3)$$

where d is the remaining size of this TCP flow.

When sending out a scheduling packet, the sender will add eight extra parameters to the header of this packet. In the eight parameters, one is the flow-level priority, one is the current desired rate, and the other six are reserved for allocating rates by at most six switches. We initialize the values of these six parameters to R.

When the sender receives a scheduling ACK, it fetches the six allocated rates and adjusts its sending rate to the minimum value of these allocated rates.

2) Switch Operations: Each switch maps a flow-level priority to a class and maintains four variables for the class at each output port. They are $Class\_id$, $Demand$, $Alloc$ and $Flow\_num$ respectively. Here, $Demand$ denotes the total desired rates of the TCP flows in the class. $Alloc$ is the total allocated rates for these flows and $Flow\_num$ is the number of flows.

The switch works as the following steps:

- Step 1. On receiving a scheduling packet, update local information and allocate a certain rate to this TCP flow:
  - 1.1. Get all rate related parameters and the flow-level priority. Get $Class\_id$ of this TCP flow.
  - 1.2. Regard the minimum value of the seven rate related parameters as the flow's desired rate. Update $Demand$ and $Flow\_num$ for this class.
  - 1.3. Compute tot_alloc by summing up all $Alloc$s of the classes which have higher priorities than this class. Subtract tot_alloc from the output link capacity to get $avail\_band$.
  - 1.4. If $avail\_band$ is less than 0, assign 0 to this flow; if $avail\_band$ is no less than $Demand$, assign the flow's desired rate to this flow; otherwise, assign $\frac{avail\_band}{Flow\_num}$ to this flow.
  - 1.5. Update $Alloc$. Add the assigned rate to the header of the scheduling packet and then deliver this packet to the next hop.
- Step 2. On receiving a scheduling ACK, update local information:
  - 2.1. Get all rate related parameters and the flow-level priority. Get the $Class\_id$ of this TCP flow.
  - 2.2. Set the assigned rate for this flow to the minimum value of the seven rate related parameters, and update $Alloc$. Then deliver this packet to the next hop.

3) Receiver Operations: When receiving a scheduling packet, the receiver fetches the eight parameters in the header, adds these parameters to the header of a scheduling ACK, and then sends out this ACK.

Our FP-MDFS can provide *work-conservation* since it allocates unused bandwidth to flows with a lower priority. In addition, FP-MDFS is *preemptive* as it can rob bandwidth from flows with a lower priority and assign to flows with a higher priority if necessary.

D. Discussion about the Scheduling Overhead

Header: The header of a scheduling packet/ACK and an advertising packet requires 48 and 33 additional bytes on average, respectively. This overhead is only a little higher than that of the decentralized schemes such as Baraat and D-CAS, which need about two or three dozen additional bytes. However, it is much lower than the centralized scheme such as Varys, which uses massive additional TCP flows to transfer the scheduling messages.

Scheduling Delay: The total scheduling delay will take at most 15 RTTs. This value is usually less than 1 millisecond in DCNs, and it is much less than that of Varys which can be 30 milliseconds.

III. Performance Evaluation

In this section, we use the trace-based simulator provided in [5] to evaluate the performance of CGM-PS by comparing it with D-CAS, Baraat, Varys and a per-flow fairness mechanism.

Trace Setting: Similar to the settings in [5] and [7], we divide all coflows into four categories based on their width and length: *Narrow&Short*, *Narrow&Long*, *Wide&Short*, and *Wide&Long*. We consider a coflow as narrow if it consists of at most 50 flows. We set the upper bound of the length of a coflow to be 1000MB. A coflow is considered to be short if its length is less than 10MB. Each type of coflows contains 52%, 16%, 15% and 17% of coflow stream respectively.

All coflows are assumed to arrive in a Poisson process with parameter λ. We set $\lambda = \frac{avgNL \times C}{avgCS}$, where $avgNL$ is the average network load, C is the network capacity and $avgCS$ denotes the average coflow size. By adjusting the average network load $avgNL$, we can simulate the scenarios with different arriving rates.

In our simulations, the network topology is abstracted into a non-blocking switch [5] which interconnects all servers. We only focus on the access link from these servers to the switch. The capacity of each access link is set to 1Gbps.

Parameter setting: In CGM-PS, we set δ to 100ms, ϕ to 0.1, N^* to 10. For D-CAS, we set $T = 1$ second, $\delta =$ 100ms and $thresholdVolume =$ 1MB. For Baraat, we set its threshold of large-coflow identifying to 80th percentile of the coflow size. We set $T = 1$ second and $\delta =$ 100ms for Varys.

A. Impact of Network Scale

In this subsection, we set the number of coflows to be 200 and the average network load to be 1. By varying the number of servers in the cluster from 20 to 100, we explore the impact of cluster/network scale on the CCT performance. Fig. 1 (a) and (b) show the average and 90^{th} percentile CCTs when the network scale changes. Fig. 1 (a) and (b) show that CGM-PS can achieve the minimal CCTs both on average and in the 90^{th} percentile. We can also see that, with the increasing of the network scale, all curves first increase to some points

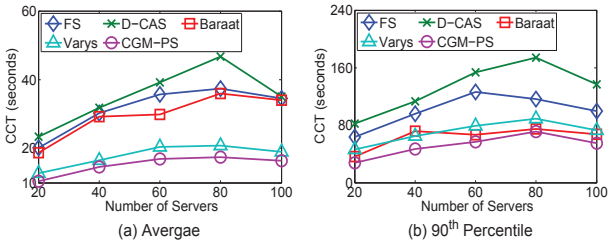

Fig. 1. CCTs when the number of servers varies from 20 to 100

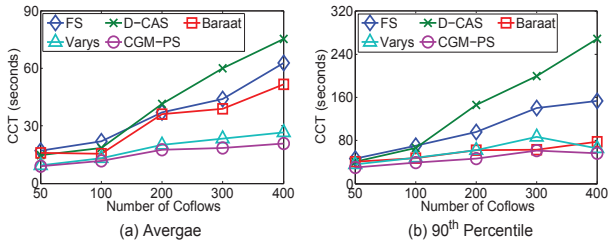

Fig. 2. CCTs when the number of coflows varies from 50 to 400

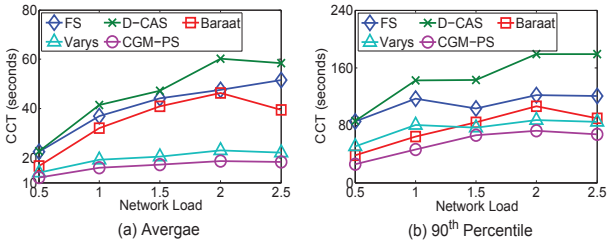

Fig. 3. CCTs when the average network load varies from 0.5 to 2.5

and then decline. This is because that when the cluster scale becomes large enough, there will be more redundant network capacity which can speed up the transferring of coflows.

B. Impact of Coflow Number

In this subsection, we set the number of servers in the cluster to 50 and the average network load to 1. We vary the number of coflows from 50 to 400 to investigate the impact of coflow number on the CCT performance. Fig. 2 (a) and (b) show the simulation results. From Fig. 2 (a) and (b) we observe that, almost all curves grow when the number of coflows increases. This is because the increasing number of coflows will decrease the available network capacity for serving a coflow on average. However, both the average and the 90^{th} percentile CCTs of CGM-PS grow slightly and their curves are always below the curves of other schemes. That means our CGM-PS works well when the number of coflows changes.

C. Impact of Network Load

In this subsection, the number of coflows is fixed to be 200 and the number of servers in the cluster is set to 50. We change the average network load from 0.5 to 2.5. Simulation results in Fig. 3 (a) and (b) show that, for all scheduling schemes, the more congestion the network is, the larger CCT will be. However, among all scheduling schemes, our CGM-PS can still achieve the best CCT performance both on average and in the 90^{th} percentile when the average network load varies.

Notice that, Varys has been proved to achieve the near-optimal CCT, but in our simulations its performance is a little worse than CGM-PS. This can be interpreted as follows. First, Varys only re-computes rates for TCP flows when a new coflow arrives at the network or a coflow departs from the network. It can't update the rate of each TCP flow in time, thus it can waste some network capacity. Second, in Varys, the method to avoid starvation is to make the un-served coflows share the network for a while every T seconds. This strategy will work poorly when the coflows' sizes are large. Third, Varys doesn't provide support to ensure the important flows of the coflows to finish data transfers preferentially when the network resource is limited.

IV. Conclusion

We proposed a novel coflow scheduling scheme named CGM-PS to shorten CCT while overcoming the weaknesses of existing coflow scheduling schemes. CGM-PS provides not only inter-coflow but also intra-coflow scheduling. For inter-coflow scheduling, we proposed the P-SEBF policy to give coflow-level priorities in a partial-decentralized manner. For intra-coflow scheduling, we proposed the FP-MDFS policy to give flow-level priorities and appropriate rates to TCP flows in a decentralized manner. We carried out comprehensive trace-based simulations and compared the CCT performance of CGM-PS with all existing scheduling schemes. Simulation results show that CGM-PS can achieve the minimal CCTs both on average and in the 90^{th} percentile. Compared with the decentralized schemes (D-CAS and Baraat), CGM-PS brings only a little higher scheduling overhead but can reduce the CCT to a fraction. Compared with the centralized scheme (Varys), CGM-PS can achieve even shorter CCT while bringing much lower scheduling overhead.

V. Acknowledgments

This work is supported in part by National Science Foundation of China (Grant No. 61303250), the Strategic Pilot Project of Chinese Academy of Sciences (Grant No. XDA06010306). The corresponding author of this paper is Yan Zhang.

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. Zaharia, M. Chowdhury, and et al., "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.

[3] D. Borthakur, "Hdfs architecture guide," *Hadoop Apache Project*, p. 53, 2008.

[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*, 2012, pp. 31–36.

[5] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 443–454.

[6] 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 SIGCOMM*, 2014, pp. 431–442.

[7] S. X. Luo, H. F. Yu, and et al., "Minimizing average coflow completion time with decentralized scheduling," in *ICC*, 2015.