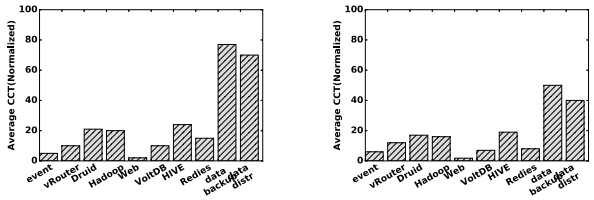
Yosemite: Efficient Scheduling of Weighted Coflows in Data Centers

Han Zhang*‡, Xingang Shi†‡, Xia Yin*‡, Zhiliang Wang†‡
*Department of Computer Science and Technology, Tsinghua University
†Institute for Network Sciences and Cyberspace, Tsinghua University
‡Tsinghua National Laboratory for Information Science and Technology (TNLIST)
Beijing, P.R. China
Email:{zhanghan, wzl, yxia}@csnet1.cs.tsinghua.edu.cn, {shixg}@cernet.edu.cn

***Abstract*—Recently, coflow has been proposed as a new abstraction to capture the communication patterns in a rich set of data parallel applications in data centers. Coflows effectively model the application-level semantics of network resource usage, so high-level optimization goals, such as reducing the transfer latency of applications, can be better achieved by taking coflows as the basic elements in network resource allocation or scheduling. Although efficient coflow scheduling methods have been studied, in this paper, we propose to schedule *weighted coflows* as a further step in this direction, where weights are used to express the emergences or priorities of different coflows or their corresponding applications. We design an information-agnostic online algorithm to dynamically schedule coflows according to their weights and the instantaneous network condition. Then We implement the algorithm in a scheduling system named Yosemite. Our evaluation results show that, compared to the latest information-agnostic coflow scheduling algorithms, Yosemite can reduce more than 40% of the WCCT (Weighted Coflow Completion Time) , and more than 30% of the completion time for coflows with above-the-average level of emergence. It even outperforms the most efficient clairvoyant coflow scheduling method by reducing around 30% WCCT, and 25%∼30% of the completion time for coflows with above-the-average emergence, respectively.**

I. Introduction

Nowadays, applications in data center need network to provide them with low latency and high throughput [5] and coflow has been proposed as a new abstraction to capture the communication patterns. According to [2], a coflow is *a collection of flows between two groups of machines with associated semantics and a collective objective.* The collection of flows share a common performance goal, e.g., minimizing the completion time of the latest flow in the collection, or ensuring that flows in the collection meet a common deadline. Coflows effectively model the application-level semantics of network resource usage, so high-level optimization goals, such as reducing the transfer latency of applications, can be better achieved by taking coflows as the basic elements in network resource allocation or scheduling. Efficient coflow scheduling methods targeting minimizing Coflow Completion Time (CCT) have been studied. Varys [2] proposes the *Smallest-Effective-Bottleneck-First* (SEBF) heuristic to determine the scheduling order of coflows and uses *Minimum-Allocation-for-Desired-Duration* (MADD) to compute the bandwidth to be allocated. However, Varys is a clairvoyant method in the sense that it depends on flow information, such as flow size or flow arrival time, to determine how to scheduling coflows. This may limit its application to scenarios where such information cannot be provided a priori. To solve this problem, non-clairvoyant methods that are flow information agnostic, such as Aalo [1], Barrat [3],and CODA [4], have also been studied. In this paper, we propose to schedule *weighted coflows* as a further step in this direction, where weights are assigned to coflows to express the emergences or priorities of different coflows or their corresponding applications. We design an information-agnostic online algorithm to dynamically schedule coflows according to their weights and the instantaneous network condition.



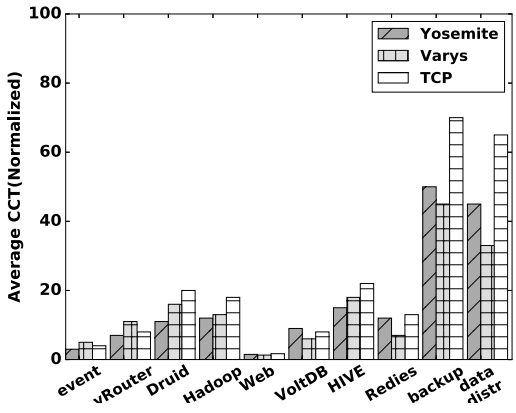
(a) Average CCT (b) Average CCT with Varys

Fig. 1: Normalized Coflow Completion Time, grouped by the level of emergence

II. Motivation

We now use real network traffic from a medium-sized data center, where more than 100 applications run on about 3000 servers simultaneously. After talking with the network operators and software development engineers about the emergence of different applications, we make an in-depth analysis of the traffic collected from 720 servers deployed in 60 racks. The traffic lasts about one month, and the basic statistics of the top ten network-demanding applications are summarized in Table I. There are typically five emergence levels, i.e., significant, important, normal, unimportant and lax. For example, the event application and the vRoute application are responsible for communication of critical components, and are of the highest level of emergence, while data-backup and data-dist are applications running in the background and are of the lowest level of emergence. To illustrate the ineffectiveness

978-1-5090-6501-1/17/$31.00



(a) Average CCT

Fig. 2: [Simulation] Average CCT (Normalized) and Average WCCT for all applications

of existing coflow scheduling methods, we plot the average Coflow Completion Time (CCT) of different coflows, grouped by their levels of emergence, in Fig.1. Fig.1(a) shows the direct measurement result of all applications, while Fig.1(b) shows the simulation result when Varys is used to schedule them. Comparing the two plots, we can see that Varys clearly reduces the completion time of less emergent coflows, such as Redies, data-backup and data-dist, at the expenses of the same of even longer completion time of more emergent coflows, such as Event and vRouter. This motivates us to optimize the Weighted Coflow Completion Time (WCCT), where weight denotes the emergence of coflows, and the more emergent the coflow, the higher the weight. Coflow scheduling algorithms should take coflow weight into consideration, and optimizing CCT becomes just a special case of optimizing WCCT.

TABLE I: Applications and their levels of emergence

App Name	Type	width	length (MB)	Emergence Level
Event	communication	20	5	significant
vRouter	communication	8	3	significant
Druid	interactive	6	18	important
Hadoop	computation	5	42	normal
Web	interactive	3	5	normal
VoltDB	background	4	21	normal
Hive	background	7	32	unimportant
Redies	background	2	30	unimportant
data-backup	background	3	124	lax
data-dist	background	5	93	lax

III. Algorithm

Efficient algorithm is needed to schedule coflows according to the network condition and emergence level. Algorithm 1 prioritizes coflows according to the heuristics explained above (line 1 $\sim$ 13) and allocates bandwidth to each flow (line 14 $\sim$ 22). At last, it distributes the unallocated bandwidth equally among all unfinished flows to provide resource conservation. Algorithm 1 is an information-agnostic online algorithm that dynamically schedule coflows according to their weights and the instantaneous network condition.

Algorithm 1 Information-agnostic Online scheduling to minimize WCCT

1: **if** a new coflow arrives **then**
2: update the coflow list $\mathcal{F} = \{F^{(k)}\}$
3: $n = |\mathcal{F}|$
4: **for** $1 \le k \le n$ **do**
5: update $s_{i,j}^{(k)}$, the cumulative volume of the traffic sent by coflow $F^{(k)}$, for $1 \le i, j \le m$
6: $L_i^{(k)} = \sum_{j=1}^{m} s_{i,j}^{(k)}$ for $1 \le i \le m$
7: $L_{j+m}^{(k)} = \sum_{i=1}^{m} s_{i,j}^{(k)}$ for $1 \le j \le m$
8: $l^{(k)} = \min_{1 \le i \le 2m} L_i^{(k)}$
9: $\alpha^{(k)} = l^{(k)} / w_k$
10: **end for**
11: ω =sort the list of $\{\alpha^{(k)}\}$ in nondecreasing order
12: γ =index set of ω
13: **end if**
14: initialize the bandwidth to be allocated on each ingress and egress port: $I_p = O_p =$ port physical bandwidth, for $1 \le p \le m$
15: **for** ℓ from 1 to n **do**
16: schedule coflow $F = F^{(\gamma[\ell])}$
17: r = min $\{I_p, O_q\}$, subject to F still need to send traffic to port p or receive traffic from port q
18: **for** each unfinished flow $F_{i,j}$ in F **do**
19: allocate bandwith of r to $F_{i,j}$
20: update I_i and O_j by deducting r from them
21: **end for**
22: **end for**
23: allocate remaining bandwidth equally to all flows

To test the performance of Yosemite, we set five emergence levels and Fig. 2 shows the result. We can see that for vRouter and event (Signiificant),Yosemite performs about 20% better than Varys. For Hadoop and Druid (Important), Yosemite performs about 30% better than Varys. For other unimportant applications, Varys performs about 10%$\sim$20% better than Yosemite.

References

[1] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 393–406. ACM, 2015.

[2] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 443–454. ACM, 2014.

[3] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 431–442. ACM, 2014.

[4] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng. Coda: Toward automatically identifying and scheduling coflows in the dark. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 160–173. ACM, 2016.

[5] H. Zhang, X. Shi, X. Yin, F. Ren, and Z. Wang. More load, more differentiationa design principle for deadline-aware congestion control. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 127–135. IEEE, 2015.