Coflourish: An SDN-Assisted Coflow Scheduling Framework for Clouds

Chui-Hui Chiu, Dipak Kumar Singh, Qingyang Wang, Seung-Jong Park
Division of Computer Science and Engineering, Center for Computation and Technology
Louisiana State University, Baton Rouge, LA, USA 70803
E-mail: {cchiu1, dsingh8, qwang26, sjpark}@lsu.edu

***Abstract*—Existing coflow scheduling frameworks effectively shorten communication time and completion time of cluster applications. However, existing frameworks only consider available bandwidth on hosts and overlook congestion in the network when making scheduling decisions. Through extensive simulations using the realistic workload probability distribution from Facebook, we observe the performance degradation of the state-of-the-art coflow scheduling framework, Varys, in the cloud environment on a shared data center network (DCN) because of the lack of network congestion information. We propose `Coflourish`, the first coflow scheduling framework that exploits the congestion feedback assistances from the software-defined-networking(SDN)-enabled switches in the networks for available bandwidth estimation. Our simulation results demonstrate that `Coflourish` outperforms Varys by up to 75.5% in terms of average coflow completion time under various workload conditions. The proposed work also reveals the potentials of integration with traffic engineering mechanisms in lower levels for further performance optimization.**

***Index Terms*—Application-aware Networks; Coflow Scheduling; Data Center Networks; Software-defined Networking; Cloud Computing**

I. Introduction

The prosperous growth of big data related researches and applications result in the thriving of cluster computing frameworks [11], which greatly simplify the development of applications for big data analysis. An application running on one of these frameworks typically consists of a sequence of dependent steps such as the Map, Shuffle, and Reduce phases in the MapReduce framework. There are data exchanges among machines between steps. Machines exchange the output data of the previous step to prepare the input data for the next step. The application can resume only when the data exchange completes. In data-intensive applications, data exchange time can contribute to as much as 70% of the application completion time [7]. Shortening the data exchange time reduces a large proportion of the application completion time.

A data exchange mostly involves multiple parallel communication flows between machines. The data exchange completes only when all associated flows finish. Strategies for shortening the data exchange time should consider all associated flows as a logic unit. Coflow [5] is proposed to abstract the collective communication characteristics such as requirement and behavior between two groups of machines. Coflow scheduling frameworks [8], [6] allow programmers to specify the communication characteristics of applications using the coflow abstraction and shorten the average coflow completion time (CCT) by coordinating coflows' transmission order and transmission rates. Evaluations show encouraging decreasing of CCT. The existing coflow scheduling frameworks such as Varys [8] are designed on the assumption that the underlying network can be seen as an ideal non-blocking switch connecting all machines which are monitored and controlled by the frameworks' cooperative daemon processes. Making scheduling decisions by considering only the available bandwidth of the network interface card (NIC) on each controlled machine is sufficiently satisfying.

However, the non-blocking switch assumption is likely to not hold when the existing coflow scheduling frameworks are deployed in a cloud environment. A cloud is typically a cluster of machines connected via a DCN [1], [13] shared by numerous tenants such as the Amazon EC2 [4] and Google Cloud Platform [12]. Tenants are free to deploy customized software environments on their machines. Thus, the existing coflow scheduling frameworks cannot guarantee that their daemons control all machines. Not controlled machines may run various applications which generate complicated communication flows across the cloud network. We refer to this category of communication flows as the background traffic in the rest of this paper. The network bandwidth consumed by the background traffic is transparent to the existing coflow scheduling frameworks. Once the background traffic is huge such as high-definition video streaming [3], the existing coflow scheduling frameworks do not perform as well as expected.

In this paper, we study the performance degradation of the existing coflow scheduling frameworks such as Varys in the cloud environment and improve the performance by exploiting feedbacks from the switches in the DCN.

We motivate ourselves by studying the performance loss resulted in by the background traffic. We implement a trace-driven simulation of the Varys. The simulation runs on the workload from the benchmark Facebook traffic probability distribution. We observe up to 82.1% decrease of CCT (II). We realize that accurate available network bandwidth information is crucial to resolve the issue.

We propose `Coflourish`, the first coflow scheduling framework which takes congestion feedbacks from SDN-enabled switches in the fat-tree-based DCN [1], [13] for more accurate estimation of available network bandwidth (III). Every switch keeps track of the congestion information of

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.10

every port. A feedback mechanism periodically aggregates the congestion information from each fat-tree layer for each host machine. A daemon process on each machine participating in the framework converts its collected congestion information into available bandwidth from(to) itself to(from) each fat-tree layer and report the available bandwidths to a logically centralized scheduler. The scheduler can easily synthesize the reported bandwidths and use the smallest bottleneck bandwidth as the overall available machine-to-machine bandwidth. `Coflourish` is the first coflow scheduling framework which provides detailed algorithms of the SDN-assisted available bandwidth estimation.

To evaluate `Coflourish`, we create a trace-driven simulation and run it on a large variety of DCN workloads in the flow level. In the presence of background traffic, simulation results show that our framework estimates the available network bandwidth 4.1% more accurately then Varys under light background traffic and 78.7% more accurately under heavy background traffic. Simulation results also reveal that our framework shortens the average CCT up to 75.5% compared to Varys (IV).

The rest of the paper is organized as follows. Section II explores the performance of the existing coflow scheduling framework in the cloud environment and reveals the motivation behind the `Coflourish`. Section III presents the architectural design of `Coflourish`. Section IV shows the evaluation results. Section V distinguishes our work from previous ones. We conclude in Section VI.

II. Motivation

To understand the impact of the network congestion to the existing coflow scheduling framework in clouds, we construct a trace-driven flow-level simulation of Varys and examine Varys' performance under various network traffic patterns.

We assume the topology of the underlying DCN to be the de-facto spine-leaf fat-tree [2] shown in Figure 1. Detailed simulation configurations are provided in IV-A. Two types of traffic loads are generated, regular size coflow traffic load (Regular) and large size coflow traffic load (Large). The Regular load represents the load which consists of traffic from regular cluster applications such as the MapReduce. The Large load represents the load which consists of 50% traffic from applications that consistently produce large traffic such as HD video streaming and 50% traffic from the regular applications. We convert a certain weight(or percentage) from generated traffic load into the background traffic by removing their coflow related attributes to impose unknown congestions in network. The weight of background traffic varies from 10% through 90% with a 10% interval. We compare the performance of Varys with an ideal scheduling framework which runs identical scheduling algorithm as Varys' but knows the precise available bandwidth of each network link when making new coflow schedules and immediately reschedules when there is any change of available bandwidth at any network link.

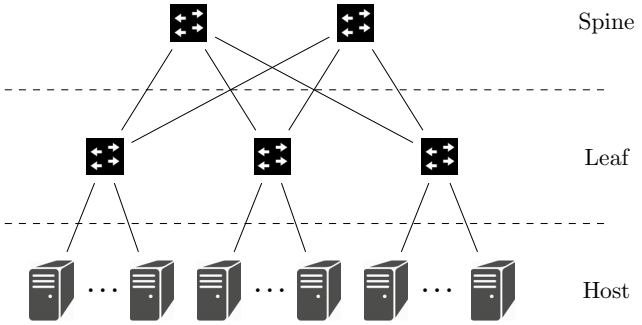


Fig. 1: A spine-leaf fat-tree DCN with 2 spine switches and 3 leaf switches. Each switch in the leaf layer connects to the same number of host machines.

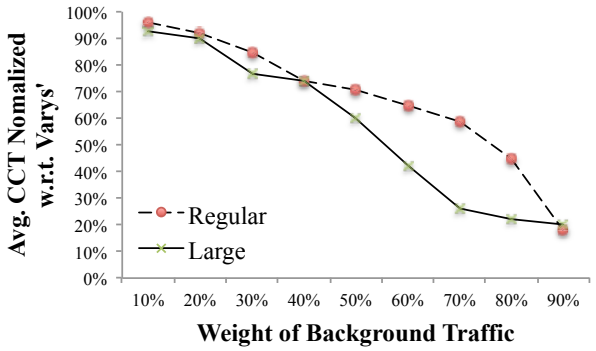


Fig. 2: CCT of the ideal scheduling framework normalized to Varys'.

The average CCTs normalized to the Varys' using Equation 1 are shown in Figure 2.

The ideal scheduling framework achieves 4.1%(82.1%) decrease in average CCT with 10%(90%) background traffic under the regular size coflow traffic load and achieves 7.3%(80.2%) decrease in average CCT with 10%(90%) background traffic under the large size coflow traffic load. By comparing the results of the Varys and the ideal scheduling framework, we conclude that accurate network congestion information is critical to an efficient coflow scheduling.

$$\text{Normalized CCT} = \frac{\text{New Fwk.'s CCT}}{\text{Varys' CCT}} \quad (1)$$

III. Framework Design

We propose a new coflow scheduling framework, `Coflourish` which inherits the preferable features of the existing framework, Varys, and alleviates the performance degradation problem resulted in by the inaccuracy of the available network bandwidth estimation in Varys. The design goals of `Coflourish` are as follows.

- **Switch-assisted Bandwidth Estimation:** Network switches in the DCN should report the instant available

network bandwidth information to the coflow scheduler for more accurate available bandwidth estimation. Complicated computations should be avoided on switches.

- **Average CCT Minimization:** The average CCT should be comparable to existing coflow scheduling frameworks.
- **Starvation Prevention:** Coflows should not be suspended for a unpredictable amount of time because of the lack of necessary resources to continue.
- **Work Conservation:** Any resource which any coflow needs to make progress should be allocated.

We first introduce the overall workflow of all system components and then individually elaborate the detailed design for each.

A. Framework Workflow

`Coflourish` requires several components in the framework, network switches, host machines, and a logically centralized coflow scheduler, to tightly coordinate to achieve its design goals. Major steps are as follows.

1) Each switch in the spine layer periodically sends per-port congestion information to each switch in the leaf layer which itself connects to. (III-B)
2) Each switch in the leaf layer, upon receiving the congestion information from the spine layer, sends its own per-port congestion information and the aggregated information received from the spine layer to the host machine which it connects to in the host layer. (III-B)
3) Each host machine in the host layer, upon receiving the congestion information from switches, calculates the available network bandwidth from itself to each switch layer using the received information or estimates any missing piece. The host machine sends the available bandwidth information to the coflow scheduler. (III-C)
4) The coflow scheduler estimates available bandwidth for each source-destination machine pair and make scheduling decisions to achieve average CCT minimization, starvation prevention, and work conservation. (III-D)

B. SDN-enabled Switch

To accurately estimate the congestion in a DCN, we design custom configuration for network links, and data structures and algorithms for switches. Table I provides the meaning of symbols used in algorithms in the rest of this paper. Figure 3 illustrates the DCN on which all algorithms operate. To save space, we omit the host machines which connect to the leaf switches other than L_n. We assume all network links have identical capacities.

For each network link, the link is sliced into two logical channels. One with higher priority exclusively transfers the congestion feedback messages from switches to ensure that critical network congestion information delivery is not delayed by other types of traffic. The other one carries the actual data packets. The slicing can easily be implemented using existing technology such as the Priority Code Point (PCP) in the Ethernet.

TABLE I: Symbols used in algorithms

Symbol	Description
C	Network link capacity.
CT	Congestion table on current switch.
CT_p	Congestion value in the CT for Port p.
$Pt(X, Y)$	Port on Switch X connecting to Switch Y.
AB(X)	Available bandwidth of Port X.
P_i^{out}	egress port on Machine i (H_i).
P_i^{in}	ingress port on H_i.
S	Set of all switches in the spine layer.
s	Total elements in S.
L	Set of all switches in the leaf layer.
ℓ	Total elements in L.
H	Set of all host machines in the host layer.
h	Total machines a leaf switch connects to.
$X_{y..z}$	Element with Index y through z in set X.
UL_i	Avl. BW to leaf layer switch from H_i.
DL_i	Avl. BW from leaf layer switch to H_i.
US_i	Avl. BW to spine layer switch from H_i.
DS_i	Avl. BW from spine layer switch to H_i.
C_{sel}	Set of coflows with BW allocation.
C_{str}	Set of coflows without BW allocation.

For each switch, the switch maintains a congestion table and algorithms for updating the table and feeding back congestion information from every switch layer to the host layer. The a spine switch initiates a feedback every T_{FB} interval.

The congestion table tracks the congestion of each switch port with only one register and is updated using the EstCong (in Algorithm 1). EstCong increments the congestion value by the size of each packet which passes through (Line 2-4) and decrements by multiplying $(1 - \alpha)$ in the range of $(0, 1)$ every T_{dre} interval (Line 5-9). This algorithm is a simplified Discontinuing Rate Estimator (DRE) [2] without computing the congestion metric. The adoption of DRE reveals a potential of `Coflourish` for integration with lower layer traffic engineering mechanisms in DCN.

The T_{FB}, α, τ, and T_{dre} are dynamically tunable by the SDN controller.

Algorithm 1 Congestion Estimation

```
1:  procedure EstCong(Event E, CT)
2:      if E is packet arrival then
3:          p ← event source switch port
4:          CT_p ← CT_p + packet size
5:      else if E is decrease timeout then
6:          for all CT_x ∈ CT do
7:              CT_x ← (1 − α)CT_x
8:          end for
9:          Schedule a decrease timeout in T_dre interval
10:     end if
11: end procedure
```

Each switch layer periodically feeds back its congestion information and the any aggregated congestion information from the next layer which is farer from the host layer to

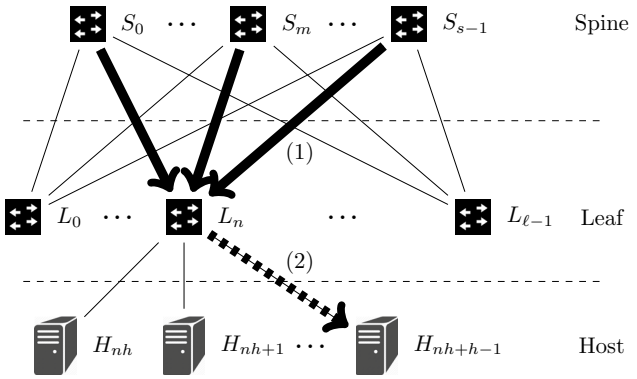


Fig. 3: Switch feedback mechanism to machine H_{nh+h-1}. Phase (1): spine layer feedbacks to leaf layer (solid arrow). Phase (2): leaf layer feedbacks to host layer (dashed arrow).

the next layer which is closer to the host layer every T_{FB} interval. In Figure 3, each switch in the spine layer feeds back congestion information to the leaf layer, marked as Stage (1), and each switch in the leaf layer feeds back congestion information to the host layer, marked as Stage (2). During Stage (1), each $S_x \in S$ runs FBSPINE (in Algorithm 2). The congestion value of each egress port which connects to Leaf Switch L_x (Line 3-4) is sent to L_x as the from-spine-to-leaf congestion. During Stage (2), each $L_x \in L$ runs AGGFBLEAF (in Algorithm 2). Each leaf switch sends three pieces of congestion information to Machine H_x. First, the congestion value of the egress port which connects the current leaf switch to H_x (Line 15-17) as the from-leaf-to-host congestion. Second, the summation of congestion values of the egress ports which connect the current leaf switch to all spine switches (Line 10-12) as the from-leaf-to-spine congestion. Third, the summation of the from-spine congestion values from all spine switches (Line 9). We only consider the congestion at egress ports because congestion happens only at egress ports. The ingress port is always not congested because the ingress traffic must be the egress traffic which is shaped by the congestion at some egress port. As revealed in the on-switch algorithms, no complex computation is involved. Major complexity is on the host machine and the scheduler. `Coflourish` imposes little overhead to switches.

To reduce the traffic load introduced by the feedback mechanism, each switch send feedback messages using simple transmission protocols such as Point-to-Point Protocol (PPP) and High-level Data Link Control (HDLC).

An SDN-enabled switch is easily be customizable to accommodate this congestion estimation mechanism. In the cloud environment, the SDN-enabled switch can be either physical or software switch on the host machine to connect virtual machines. For example, a physical switch which support the P4 programming language [10] or the Open vSwitch software switch which is already widely deployed in production cloud management systems [9].

Algorithm 2 Congestion Feedback & Aggregation

```
1:  procedure FBSPINE(CT, L)
2:      for all L_x ∈ L do
3:          pe ← Pt(current switch, L_x)
4:          fS ← CT_pe                          ▷ S to L cong.
5:          Send fS to L_x
6:      end for
7:  end procedure

8:  procedure AGGFBLEAF(Feedbacks FB, CT, S, H)
9:      fS ← ∑(fS ∈ FB)                         ▷ Agg. S to L cong.
10:     for all S_x ∈ S do
11:         pe ← Pt(current switch, S_x)
12:         tS ← ∑ CT_pe                        ▷ Agg. L to S cong.
13:     end for
14:     n ← current leaf switch ID
15:     for all H_x ∈ H_{nh..nh+h-1} do
16:         pe ← P(current switch, H_x)
17:         fL ← CT_pe                          ▷ Agg. L to H cong.
18:         Send fS, tS, fL to H_x
19:     end for
20: end procedure
```

C. Host Machine

The host machine in `Coflourish` does two tasks, finding the smallest bottleneck available bandwidth from(to) itself to(from) each switch layer for later bandwidth estimation at the scheduler, and pacing the transmission of flows in coflows using the transmission rate given by the scheduler. In this work, we only consider physical machines and switches in one DCN for easy explanations, but algorithms can be further extended to virtualized environment.

For finding the bottleneck available bandwidth, the host machine runs AGGRPTHOST (in Algorithm 3). According to [2], the congestion value (Cng) is proportional to the bandwidth of traffic. The available bandwidth (A) of a link can be derived using Equation 2.

$$Cng = \lambda(C - A)\tau \Rightarrow \lambda\tau = \frac{Cng}{C - A} \Rightarrow A = C - \frac{Cng}{\beta} \quad (2)$$

where C is the capacity of the link, λ is a scaling factor, τ is a time constant, and $\beta = \lambda\tau$ is a scaling factor. The from-leaf-to-host congestion value is fed back from leaf layer to the host machine (fS in Line 5). The host machine also knows its own exact available bandwidth from the leaf switch to itself (Line 4). Thus, the β in Equation 2 is calculated (Line 6). The leaf-to-spine available bandwidth is calculated (Line 7). The tS is an aggregated congestion value of all network links from a leaf switch to all s spine switches so that the link capacity used in Equation 2 should also be aggregated as sC. The bottleneck available bandwidth from current machine to spine layer is calculated (Line 8). The bottleneck available bandwidth from spine layer to current machine can be calculated through a similar process (Line 9-10). Finally, The available bottleneck bandwidths from current machine to each switch layer are

reported to the Coflourish scheduler using a reliable transmission protocol such as TCP.

Algorithm 3 Congestion Aggregation & Report

1: **procedure** AGGRPTHOST(Feedback FB)
2: $i \leftarrow$ current machine ID
3: $UL_i \leftarrow$ AB(P_i^{out}) ▷ BW to L.
4: $DL_i \leftarrow$ AB(P_i^{in}) ▷ BW from L.
5: fS, tS, $fL \leftarrow fS$, tS, $fL \in FB$
6: $\beta \leftarrow fL$ / $(C - DL_i)$
7: $LtoSBW \leftarrow sC - \frac{tS}{\beta}$
8: $US_i \leftarrow \min(LtoSBW, UL_i)$ ▷ BW to S.
9: $StoLBW \leftarrow sC - \frac{fS}{\beta}$
10: $DS_i \leftarrow \min(StoLBW, DL_i)$ ▷ BW from S.
11: Send US_i, DS_i, UL_i, DL_i to scheduler
12: **end procedure**

For pacing the transmission of flows in coflows, the cooperative daemon process of Coflourish receives transmission rate of each egress communication flow from the scheduler and throttles each flow using the mechanism similar to Varys'.

The report interval and the number of aggregated links, s, can be dynamically changed by the SDN controller.

D. Coflow Scheduler

The scheduler of Coflourish decides the order in which coflows receive network bandwidth allocations and the amount of allocated bandwidth at which each flow in each coflow sends data to achieve all design goals. We enhance the state-of-the-art existing algorithm in Varys since many of its design goals are identical to ours. In this section, we briefly cover the complete logics of the scheduler and put emphasis on our enhancements. The logic which achieves each design goal is highlighted. The scheduler is also the SDN controller. We explain in details at the end of this section.

To provide switch-assisted bandwidth estimation, we add a data structure, PathRem, shown in Table II to help the scheduler trace the bi-directional available bottleneck bandwidth from(to) each host machine to(from) each switch layer. The PathRem synthesizes all the available bottleneck bandwidth information sent from the daemon on each host machine. PathRem(i,j,h) represents the smallest available bottleneck bandwidth from the source host Machine i to the destination host Machine j. Equation 3 illustrates the algorithm used by the PathRem. PathRem (1) calculates the rendezvous switch layer of the traffic from the source machine (up direction) and the traffic to the destination machine (down direction); (2) retrieves the estimated available upward and downward bandwidths; (3) calculates the overall available bottleneck bandwidth along the path across the DCN.

TABLE II: The PathRem which stores bi-directional available bottleneck bandwidths at the scheduler.

Direction	Layer	Host			
		H_0	H_1	...	$H_{\ell h-1}$
Up	Leaf	UL_1	UL_2	...	$UL_{\ell h-1}$
	Spine	US_1	US_2	...	$US_{\ell h-1}$
Down	Leaf	DL_1	DL_2	...	$DL_{\ell h-1}$
	Spine	DS_1	DS_2	...	$DS_{\ell h-1}$

$$PathRem(i,j,h) = \begin{cases} \min(UL_i,\ DL_j) & \text{if } (i \textbf{ div } h) = (j \textbf{ div } h). \\ \min(US_i,\ DS_j) & \text{otherwise.} \end{cases}$$

where **div** is the integer division operator. (3)

Current colfows are scheduled by Algorithm 4, Varys' algorithm with our enhancements. Only coflows with size greater than 25MB are scheduled for scalability concerns as suggested by Varys. Whenever a coflow arrives/finishes or the variation of the available bandwidth in the DCN exceeds a certain threshold, SERVECOFLOW is invoked and generates a new schedule. SERVECOFLOW contains two stages, CCT minimization and starvation prevention. In the CCT minimization stage, the design goals of average CCT minimization and work conservation are achieved.

To minimize the average CCT, coflows are selected to progress (Line 22) with the Smallest Effective Bottleneck First (SEBF) heuristic (Line 3). The SEBF in Varys only uses the estimated available bandwidth of the NICs as the end-to-end available bandwidth across DCN to calculate the Effective Bottleneck (EB) for each coflow. This simple method incurs huge error at the presence of the background traffic.

We improve the accuracy of the end-to-end available bandwidth estimation by introducing Equation 3. In Case 1, both ends attach to the same leaf switch so that the end-to-end available bandwidth is the minimum of the available bandwidth to the leaf layer from Machine i (UL_i) and the available bandwidth from the leaf layer to Machine j (DL_j). In Case 2, two ends attach to different leaf switch connected via the spine layer so that the end-to-end available bandwidth is the minimum of the available bandwidth to the spine layer from Machine i (US_i) and the available bandwidth to from the spine layer to Machine j (DS_j). We develop Equation 4 to determine the longest CCT (Γ) as the bottleneck of a coflow by taking the accurate end-to-end available bandwidth into consideration. We schedule coflows using the shortest bottleneck first heuristic (SBF).

$$\Gamma = \max_i(\max_j \frac{d_{ij}}{PathRem(i,j,h)}) \quad (4)$$

where d_{XY} is the summation of the remaining data size of the current coflow from Machine X to Machine Y.

To provide work conservation, we distribute the unallocated bandwidth to the coflows which are able to progress

in the latest schedule (Line 14). In the starvation prevention stage, the design goal of starvation prevention is achieved. All coflows which cannot acquire any network bandwidth to progress in the former stage receive bandwidth allocations (Line 24-31).

To prevent starvation, the CCT minimization stage and starvation prevention stage execute in turns. The former executes for T_{sel} interval, and the latter executes for T_{str} interval. This strategy guarantees that each coflow progresses for at least T_{str} interval every $T_{sel} + T_{str}$ interval.

To effectively react to available bandwidth change due to the variation of the background traffic, we add a branch in SERVECOFLOW (Line 32-35). If the variation of the available bandwidth exceeds a tunable threshold, $thld_{BW}$, the variation is considered significant, and the current schedule is considered sub-optimal and a new schedule is generated. $thld_{BW}$ determines the sensitivity of `Coflourish` to available bandwidth change in the DCN.

Being the SDN controller as well, the scheduler is able to dynamically manipulate the behaviors of the congestion feedback mechanism on switches and the available bandwidth calculations on hosts as stated in previous sections. The `Coflourish` can dynamically adjust its performance as needed. Section IV-D illustrates the impact of some adjustments.

Algorithm 4 Scheduling with Switch-assisted BW Estimation

```
1:  procedure SHAREBW(Coflows γ, PathRem(.))
2:      for all coflow C in γ do
3:          Calculate Γ using Eq 4
4:          for all flow in C do
5:              rate ← (flow's remaining size) / Γ
6:              Update PathRem(src host,dst,h) with rate
7:          end for
8:      end for
9:  end procedure

10: function SELECT(Coflows γ, PathRem(.))
11:     C_sel = Sort all Coflows in γ in SBF order
12:     SHAREBW(C_sel, PathRem(.))
13:     Assign remaining BW to coflows in C_sel
14:     C_str = starved coflows in C_sel
15:     return C_sel, C_str
16: end function

17: procedure SERVECOFLOW(Coflows γ, PathRem(.))
18:     if is CCT min. stage then
19:                                          ▷ CCT min. stage
20:         Stop coflows in C_str
21:         C_sel, C_str = SELECT(γ, PathRem(.))
22:         Switch to feeding starved stage in T_sel interval
23:     else if is starvation prev. stage then
24:                                  ▷ starvation prev. stage
25:         Stop coflows in C_sel
26:         for all coflows in C_str do
27:             Add all flows to one_coflow
28:             SHAREBW(one_coflow, PathRem(.))
29:             Switch to min CCT stage in T_str interval
30:         end for
31:     else if net avl. BW change > thld_BW then
32:                              ▷ Significant avl. BW change
33:         C_sel, C_str = SELECT(γ, PathRem(.))
34:         go back to interrupted point of execution
35:     end if
36: end procedure
```

IV. EVALUATION

We extend the trace-driven flow-level simulation created in II to evaluate the performance of `Coflourish`. Each result is an average of 5 runs.

A. Simulation Environment

Our simulation is similar to that in [8], [14]. Coflow arrival events are inserted according to the workload trace and sorted by time in advance. So are the events of periodical congestion feedback and host report. When processing the current event, the simulation generates future events, updates state variables, or re-calculates flow remaining size according to `Coflourish`'s algorithms until no event exists. The simulation workload consists of two types of coflows: regular coflows and large coflows. A regular coflow represents the traffic load which follows the benchmark Facebook traffic probability distribution used in Varys [8](Figure 4). A large coflow is a coflow with its size uniformly distributed between 25MB and 1GB. In both Varys and `Coflourish`, only large coflows will be scheduled by the coflow scheduling framework to reduce the scheduling overhead. We vary the percentage of large coflows in the overall traffic load to evaluate the impact of large coflows on the overall coflow performance (e.g., CCT). Concretely speaking, we test three representative percentages of large coflows: 25%, 50% and 75% in our simulation. Once the overall traffic load is generated, we convert a certain weight (or percentage) from the overall load into the background traffic by removing their coflow related attributes. The purpose is to impose different levels of unknown congestions in network. The weight of background traffic varies from 10% through 90% with a 10% interval. We generate 3000 host machines connected by a spine-leaf fat-tree DCN. Each leaf switch connects to 50 host machines. The oversubscription is 10:1. The total number of coflows in our traffic load is 1000. The coflow arrival is a Poisson process. Other default framework related parameter configuration is presented in Table III.

B. Improvement of Coflourish

We evaluate Coflourish's improvement with two criteria, available bandwidth estimation accuracy and average coflow completion time.

First, we compare the available bandwidth estimation accuracies. We use regular traffic load and vary the weight of

TABLE III: Default Framework Parameter Configuration

Parameter	Value
T_{FB}	200 milliseconds.
T_{sel}	2 seconds.
T_{str}	200 milliseconds.
$thld_{BW}$	1/8 spine layer network link capacity.
τ	500 microseconds.
α	0.5.
T_{dre}	250 microseconds.

background traffic. The metric, Normalized Inaccuracy (lower is better), is defined in Equation 5.

$$\text{Normalized Inaccuracy} = \frac{|\text{New Fwk.'s Est. Err.}|}{|\text{Varys' Est. Err.}|} \quad (5)$$

where the Err. is the difference between the estimated available bandwidth by a framework and the true available bandwidth in the network. The result is shown in Figure 4a. The `Coflourish`'s estimation is 4.1% (10% background) through 78.7% (90% background) more accurate than the Varys'. When the weight of background traffic is small, the interference from the background traffic to the Varys is small so that the Varys can estimate available bandwidth with slight error using its simple NIC-based estimation. Varys' estimation is slightly worse than the estimation of `Coflourish` based on accurate available bandwidth feedback information. When the weight of background traffic is large, the interference from the background traffic to the Varys is severe so that the simple bandwidth estimation of Varys gives largely biased result. Varys' estimation is much more erroneous than the estimation of `Coflourish` base on accurate available bandwidth feedback information.

Second, we compare the average CCT. We use regular traffic load and vary the weight of background traffic. The result is shown in Figure 4b. The `Coflourish` shortens the average CCT by 4.3% (10% background) through 75.5% (90% background) compared to Varys'. As explained in the previous analysis, Varys estimates bandwidth with slight error when encountering small amount of background traffic, and Varys estimates bandwidth with huge error when encountering large amount of background traffic. Thus, Varys generates a comparable coflow schedule to `Coflourish`'s when background traffic is light, and Varys generates a much worse schedule than the `Coflourish`'s when background traffic is heavy.

C. Impact of Coflow Size

We generate 3 traffic loads with 25%, 50%, and 75% large coflows. The weight of background traffic varies from 10% through 90% with a 10% interval. The result is shown in Figure 5a.

With less than 30% of background traffic in network, background traffic impact Varys' bandwidth estimation little. Varys still generates comparable schedule to `Coflourish`.

With 30% through 80% of background traffic in network, background traffic heavily interferes the bandwidth estimation of Varys. Average CCT increases much. In contrast, `Coflourish` still generates good schedule with available bandwidth feedback information. The more the large background traffic, the larger the average CCTs differ between the Varys' schedule and `Coflourish`'s schedule.

With more than 80% of background traffic in network, the background traffic still heavily impacts Varys bandwidth estimation, but the network tends to be saturated. The gap of average CCT resulted in by different quality of schedule closes. The average CCT also stops decreasing.

D. Impact of Feedback Interval and Bandwidth Variation Threshold

We probe the impact of two critical parameters in Coflourish, the feedback interval (T_{FB}) and the bandwidth variation threshold ($thld_{BW}$).

We set the T_{FB} to 100, 200, and 400ms. We vary the weight of background traffic. The result is shown in Figure 5b. A smaller T_{FB} results in more frequent update of available bandwidth information. The up-to-date available bandwidth information results in a schedule which is closer to the optimal. A larger T_{FB} results in the opposite.

We substitute 1/4, 1/8, and 1/16 the spine layer network link capacity into $thld_{BW}$. We vary the weight of background traffic. The result is shown in Figure 5c. A smaller $thld_{BW}$ results in more frequent rescheduling with the lastest available bandwidth. Scheduling with up-to-date available bandwidth results in a schedule which is closer to the optimal. A larger $thld_{BW}$ results in the opposite.

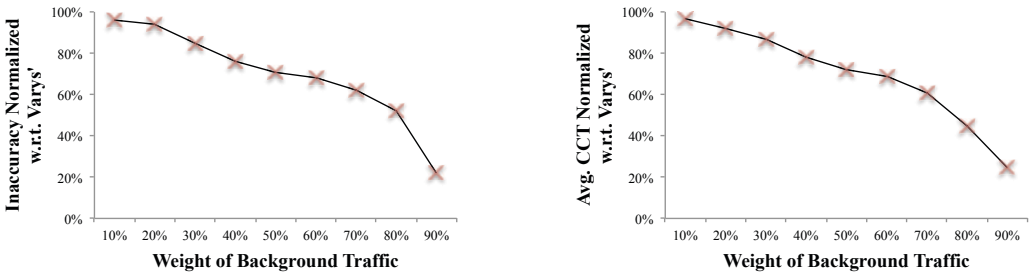
V. Related Work

Coflow Scheduling Coflow scheduling frameworks Varys [8] is the most representative work. The Varys' scheduler runs the SEBF heuristic algorithm to shorten the average CCT. Its network model is a non-blocking switch connecting all host machines, and every machine is equipped with Varys' daemon for coordination. Rapier [14] enhances Varys by taking the routing into consideration while scheduling coflows. The mechanism which feeds routing and network utilization information to the scheduler is assumed to exist.

`Coflourish` removes the strong assumption and requirement for the underlying DCN and proposes a mechanism which estimates available network bandwidth. `Coflourish`-based framework can smoothly co-exist with background traffic from not coordinated applications in a cloud environment.

Load Balancing The load balancing frameworks evenly distribute traffic among available network links. CONGA [2] maintains the leaf-to-leaf switch congestion information in the spine-leaf fat-tree DCN and place flows on the least congested links.

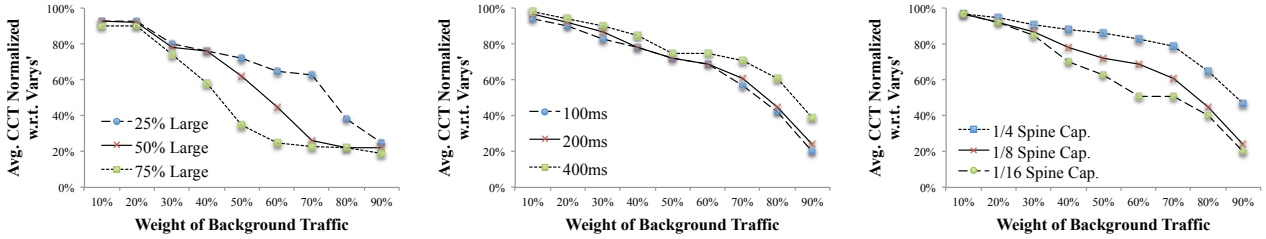
Without coflow information, load balancing frameworks consider flow as the unit for balancing and may result in sub-optimal coflow completion time. However, CONGA develops the DRE for estimating the congestion at each port. We adapt



(a) Inaccuracy

(b) CCT

Fig. 4: Improvement of Coflourish normalized to Varys.



(a) Impact of coflow size.

(b) Impact of BW Feedback Interval.

(c) Impact of BW variation threshold.

Fig. 5: Coflourish with various configurations.

its DRE for the switch-assisted bandwidth estimation function of `Coflourish`.

VI. Conclusion

Existing coflow scheduling frameworks suffer from performance degradation in a shared network environment such as the cloud because of their unawareness of the background traffic. We propose `Coflourish`, the first coflow scheduling framework which exploits congestion feedback assistances from SDN-enabled switches in the networks for available bandwidth estimation, to alleviate the problem resulted in by insufficient network information. Simulation results reveal that our framework is 78.7% more accurate in terms of bandwidth estimation and 75.5% better in terms of average CCT than the existing framework. We also discover the potential for `Coflourish` to integrate with congestion-aware load balancing frameworks in the underlying DCN.

Acknowledgment

This work was partially funded by NIH LBRN grant (P20GM103424), NSF CC-NIE award (#1341008), NSF MRI grant (MRI-1338051), NSF IBSS-L (#1620451), NSF CISE grant CNS-1566443, Louisiana Board of Regents under grant LEQSF(2016-19)-RD-A-08, and LEQSF(2015-18)-RD-A-11.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[2] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 503–514. ACM, 2014.

[3] Amazon.com. *Netflix Case Study*, 2016.

[4] Amazon.com. *Amazon Elastic Compute Cloud*, 2017.

[5] M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36. ACM, 2012.

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

[7] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 98–109. ACM, 2011.

[8] 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.

[9] R. C. Computing. *OpenStack*, 2017.

[10] P. L. Consortium. *P4*, 2017.

[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[12] Google. *Google Cloud Platform*, 2017.

[13] 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 *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[14] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 424–432. IEEE, 2015.