Exploiting Inter-Flow Relationship for Coflow Placement in Datacenters

Xin Sunny Huang, T. S. Eugene Ng
Rice University

ABSTRACT

A crucial challenge for data-parallel clusters is achieving high application-level communication efficiency for structured traffic flows (a.k.a. Coflows) from distributed data processing applications. A range of recent works focus on designing network scheduling algorithms with *predetermined* Coflow placement, i.e. the endpoints of subflows within a Coflow are preset. However, the underlying Coflow placement problem and its decisive impact on scheduling efficiency have long been overlooked.

It is hard to find good placements for Coflows. At the intra-Coflow level, constituent flows are related and therefore their placement decisions are dependent. Thus, strategies extended from flow-by-flow placement is suboptimal due to negligence of the inter-flow relationship in a Coflow. At the inter-Coflow level, placing a new Coflow may introduce contentions with existing Coflows, which changes communication efficiency. This paper is the first to study the Coflow placement problem with careful considerations of the inter-flow relationship in Coflows. We formulate the Coflow placement problem and propose a Coflow placement algorithm. Under realistic traffic in various settings, our algorithm reduces the average completion time for Coflows by up to 26%.

CCS CONCEPTS

• **Networks** → **Network resources allocation**; **Data center networks**;

KEYWORDS

Cluster schedulers; Structured traffic flow; Data center;

ACM ISBN 978-1-4503-5244-4/17/08...$15.00
https://doi.org/10.1145/3106989.3107004

ACM Reference format:
Xin Sunny Huang, T. S. Eugene Ng. 2017. Exploiting Inter-Flow Relationship for Coflow Placement in Datacenters. In *Proceedings of APNET '17, Hong Kong, China, August 3–4, 2017,* 7 pages.
https://doi.org/10.1145/3106989.3107004

1 INTRODUCTION

Achieving high efficiency in servicing structured traffic flows called Coflows [4] is a crucial challenge in modern data-parallel clusters. A Coflow represents a collection of flows between two groups of machines, and a Coflow's performance depends on its slowest flow. Coflow aware scheduling benefits distributed data processing applications by avoiding straggler [5, 6] and improving resource utilization [8].

A growing body of recent work [5, 6, 10, 16] mainly focus on optimizing the network scheduling algorithm to improve Coflows' performance. They assume *predetermined* Coflow placement, i.e. the endpoint locations of a Coflow are preset. However, a Coflow's placement can be more flexible in practice. For example, a cluster scheduler usually can choose among multiple machines to place the tasks in a newly submitted job, or the tasks for the successor stage in a staged job. Similarly, a user can also choose among multiple nodes with data copies to read from in a distributed storage system. These placement decisions would impose decisive impact on Coflows' performance (§ 3), but the underlying Coflow placement problem has long been overlooked.

In this paper, we set out to explore the Coflow placement problem. Coflow placement is much more challenging due to the inter-flow relationship in a Coflow. For example, in a one-to-many (or many-to-one) Coflow, all constituent flows share the same sender (or receiver) location. In a many-to-many Coflow, the relationship is even more complex because any member flow shares its two endpoints with two different groups of flows respectively. Therefore, placement decisions for flows in a Coflow are dependent, i.e. a flow's placement may decide the placement for other flows. As a result, negligence of such inter-flow relationship would result in poor placement decisions for certain member flows and delay the completion of the whole Coflow (§ 3.1).

To begin addressing these challenges, we propose **2D-Placement**, the first placement algorithm for Coflows that considers Coflow's inter-flow relationship. Our algorithm optimizes Coflow placement at two levels. At the intra-Coflow level, 2D-Placement leverages the inter-flow relationship to identify critical endpoints that require better placement. At the inter-Coflow level, 2D-Placement mitigates Coflows' contentions on their critical endpoints. Under realistic traffic, 2D-Placement improves the average Coflow completion time by up to 26% when compared with the state-of-the-art placement algorithm.

2 PROBLEM FORMULATION

We start by describing the conceptual models used to study the Coflow placement problem.

Coflow: A Coflow represents a collection of independent flows that share a common performance goal. A Coflow is defined by the endpoints and byte size of each flow within the Coflow. The traffic demand of a Coflow can be expressed with a matrix D, where $d_{i,j} \in D$ indicates flow $f_{i,j}$ transfers $d_{i,j}$ amount of data from sender s_i to receiver r_j.

Network Model: Topology designs such as Fat-tree or Clos [3, 9] enable building a network with full bisection bandwidth in datacenters. We model the network fabric as one non-blocking N-port switch with link bandwidth B, as assumed in previous studies on Coflow [5, 6, 10]. Switch ports are connected to *nodes*, which can be host machines or ToR switches. Under this model, only edge links are congested and the core is congestion free.

Scheduling objective: We follow the standard objectives used in studies on Coflow scheduling. At the intra-Coflow level, the objective is to minimize the Coflow Completion Time (CCT), which is the duration to finish all flows in a Coflow to speed up application level performance. For a Coflow C, denote its arrival time as t^{Arr}, and the finish time of the flow $f_{i,j} \in C$ as $t^F_{i,j}$. Hence CCT is defined as $\max\limits_{f_{i,j} \in C}(t^F_{i,j} - t^{\text{Arr}})$. At the inter-Coflow level, the objective is to minimize the sum of CCTs for a set of Coflows $\{C_1, ..., C_K\}$, so as to reduce the average latency at the application level, i.e.

$$\min \sum_{k=1}^{K}(CCT_k). \tag{1}$$

Problem statement: In an online scenario where K Coflows arrive at various time $t_1^{Arr} < ... < t_K^{Arr}$, we want to decide the placement for each Coflow C_k, which is the mapping from C_k's sender s_i (receiver r_j) to the node behind one of the input (output) ports, so that our objective Equation (1) is achieved. The placement of a Coflow C_k can be represented by mapping functions $P_k^s : \{s_i\} \rightarrow \{in.1, ..., in.N\}$ for senders and $P_k^r : \{r_j\} \rightarrow \{out.1, ..., out.N\}$ for receivers. For a specific placement, Coflow C_k's demand matrix D_k can be transformed to D'_k by setting $d'_{P_k^s(s_i), P_k^r(r_j)} = d_{s_i, r_j}$, so that $d'_{i,j}$ represents the amount of data requested by C_k from input port $in.i$ to output port $out.j$.

We assume when a Coflow arrives, its traffic demand D is available, which can be obtained in many ways such as user configurations or history of recurring jobs [12]. We do not impose assumptions on the arrival time or the traffic demand of newly arriving Coflows. Thus, our problem becomes deciding the placement of a new Coflow C_K given the existing $(K-1)$ Coflows, so that the sum of K Coflows' CCTs is minimized.

Problem analysis: The sum of CCTs is jointly determined by Coflows' placement and the network scheduling during runtime. First, Coflows' placement decides the optimal sum of CCTs achievable by *any* network scheduling policy. Later in this paper, we will show how Coflows' placement changes this optimum (§ 3). Second, after Coflows are placed, the sum of CCTs will be further determined by the network scheduling policy, which arbitrates bandwidth allocation for each Coflow.

In this paper, we focus on finding Coflow placement that minimizes the sum of CCTs under the *optimal* network scheduling, so as to optimize placement in a general case independent of the network scheduler. We do not have a conclusion on the hardness of such a placement problem. Given a specific placement, finding the optimal scheduling policy to minimize the sum of CCTs is NP-hard [6, 13].

3 MOTIVATING EXAMPLES

This section presents two motivating examples to provide insights for our placement problem. To facilitate discussion, we begin by introducing the methodology and useful terms in our analysis.

Optimal Coflow priority: Given a specific placement, the optimal scheduling policy is shown to be equivalent to servicing Coflows in the optimal priority order [13]. Specifically, each Coflow is assigned a unique priority and lower-priority Coflows cannot finish sooner by delaying a higher-priority Coflow. In our examples, we enumerate all possible priority orderings and show the CCTs derived from the best ordering that minimizes the average CCT under a specific placement. As we will show in our examples, Coflows' placement decides the performance upperbound for the network scheduler.

Demand bottleneck: To capture the key characteristic of Coflow demand D, we leverage a useful metric, the *bottleneck*. For a given traffic demand matrix D,

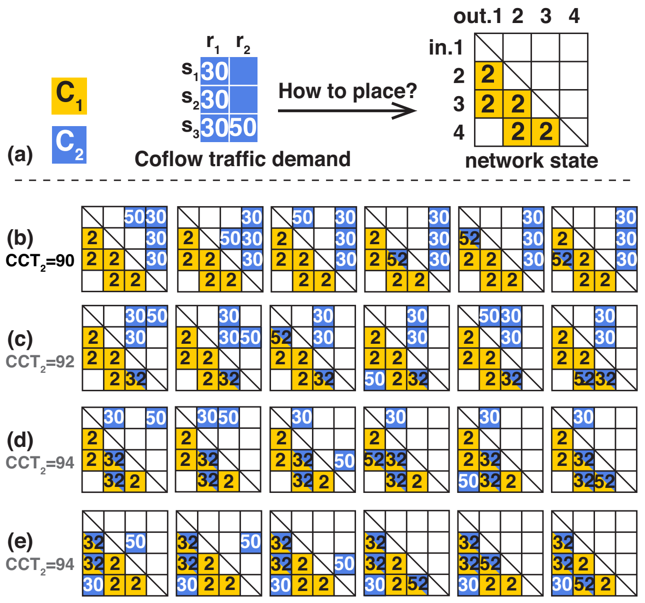

Figure 1: Coflow placement should avoid delaying bottleneck endpoint. (a) Placing 3-by-2 C_2 onto a 4-by-4 network with active flows from 3-by-3 C_1. (b-e) All possible placements of C_2. Cells with two colors indicate links shared by two Coflows. Cell numbers are the aggregated traffic size on the link. The optimal priority order is C_1, C_2. Thus CCT_1 is insensitive to C_2 placement, so only CCT_2 is considered. (b) Optimal placement with C_2's bottleneck r_1 on the least congested *out*.4. (c) Suboptimal placement due to delay of C_2's bottleneck r_1 on *out*.3 (d) Suboptimal placement due to delay of C_2's bottleneck r_1 on *out*.2 (e) Suboptimal placement due to delay of C_2's bottleneck r_1 on *out*.1. For fair comparison, we assume all Coflow traffic should traverse the network.

we define its bottleneck as $L(D) = \max(\max_j \sum_{i=1}^{N_1} d_{i,j}, \max_i \sum_{j=1}^{N_2} d_{i,j})$, where N_1 and N_2 is the number of rows and columns in D respectively. L is maximum of column sums and row sums of D. For example, in Figure 1, the bottlneck of C_2's demand matrix is 90 at r_1. Ideally when there is only one Coflow in the network, $L(D)/B$ (where B is the link bandwidth) is the lowerbound of CCT for all requested traffic to traverse the network.

3.1 Intra-Coflow Bottleneck Delay

Ideally when the available bandwidth for the newly arriving Coflow is uniform across the entire network, the Coflow's CCT is less sensitive to placement and mainly depends on the bottleneck of its demand matrix. In practice, however, the available bandwidth for the new Coflow is usually not uniform due to heterogeneous workload or reserved bandwidth for important workload. In this case, we must avoid placing a Coflow's bottleneck sender (or receiver), i.e. endpoint that gives the bottleneck, onto a port with insufficient inbound (or outbound) bandwidth, because such bottleneck endpoint(s) is more likely to delay CCT than the other non-bottleneck endpoints. For example, in Figure 1, we should place C_2's bottleneck endpoint r_1 on output port *out*.4 (Figure 1b), because higher priority C_1 already claims outbound bandwidth from *out*.1, *out*.2, and *out*.3. In other words, none of *out*.1, *out*.2 or *out*.3 provides as much available outbound bandwidth as *out*.4, and thus placing C_2's bottleneck endpoint on any of these busier ports (Figure 1c,d,e) would *delay* C_2's bottleneck and extend CCT.

In contrast, placing the *non-bottleneck* r_2 on the busier ports is *less harmful*, as shown in various solutions in Figure 1b, because non-bottlenecks can tolerate certain amount of delay before they exceed the bottleneck to effectively increase CCT. Note that in Figure 1, C_1 is prioritized under the optimal priority ordering, and thus CCT_1 is insensitive to C_2's placement. To minimize the sum of CCTs, only CCT_2 is concerned.

Hence, we have *Observation 1: When only the CCT of the Coflow to be placed is concerned, the Coflow's placement should avoid delaying the bottleneck.* To achieve this goal, bottleneck endpoint(s) should be placed at ports with sufficient bandwidth resource.

It is interesting to note that, without considering a Coflow's bottleneck, flow-level placement strategy may be suboptimal for Coflow placement. For example, prior work proposes to place a Coflow's constituent flows sequentially in the decreasing order of their flow sizes using a flow-level placement algorithm, because "large flows are more likely to be the critical flows to determine CCT" [14]. However, such strategy would yield suboptimal solutions, as shown in the first column of Figure 1c and Figure 1d. Under this flow-level strategy, the non-critical $f_{3,2}$, despite its largest flow size, takes over *out*.4, leaving suboptimal ports of *out*.2 or *out*.3 for the bottleneck receiver r_2 of C_2.

3.2 Inter-Coflow Bottleneck Contention

So far we have considered an easier case in § 3.1 where only *one* Coflow is concerned in placement. While the optimal placement in this easier case is non-trival to find, our problem would become even more challenging when the new Coflow introduces contention among multiple Coflows and thus changes *multiple* Coflows' CCTs.

Consider Figure 2 that follows our previous example. Assume shortly after we adopt the optimal placement of C_2, an incast Coflow C_3 arrives. None of the output ports is an easy choice for the new Coflow: placing C_3 on *out*.1 introduces contention with C_1; placing C_3 on *out*.2 or *out*.3 results in contentions with both C_1 and

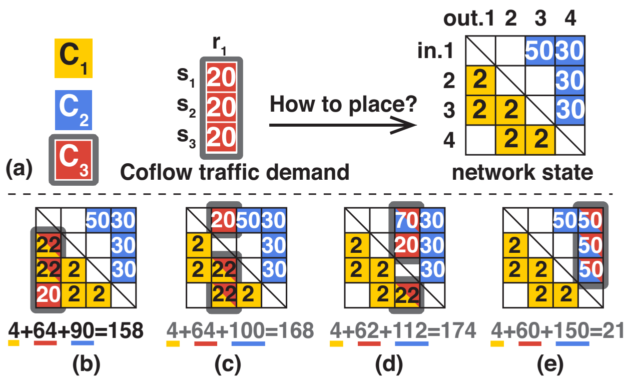

Figure 2: Coflow placement should avoid contentions among critical endpoints with heavy traffic load. (a) Placing incast C_3 onto a 4-by-4 network with active flows from C_1 and C_2. Optimal priority order is C_1, C_3, C_2. (b-e) All possible placements of C_3. CCTs are presented in $CCT_1+CCT_3+CCT_2$. (b) Optimal placement, placing C_3 on *out*.1. (c) Suboptimal placement, delaying C_2 on *in*.1. (d) Suboptimal placement, delaying C_2 on *out*.3. (e) Suboptimal placement, delaying C_2 on *out*.4. Legends and assumptions follow Figure 1.

C_2; and placing C_3 on *out*.4 leads to contention with C_2. The optimal placement is *out*.1 (Figure 2b). In Figure 2e, *out*.4 is suboptimal due to contentions among the heavy bottlenecks of C_3 and C_2. In Figure 2c, *out*.2 is suboptimal because C_3 delays C_2's non-bottleneck endpoint s_1 and prolongs CCT_2. Similarly, *out*.3 is suboptimal due to delay on r_1 of C_2, as shown in Figure 2d.

It is interesting to compare the suboptimal solutions in Figure 2c and Figure 2d. Although both solutions introduce contentions among all Coflows, however, Figure 2c is slightly better even though it involves contentions with C_1's bottlenecks. We observe contentions with C_1 is less harmful because C_1's traffic is relatively light and the resulting delay in C_3 is less dominant in the overall CCT. In contrast, contentions with C_2 should be treated more carefully due to the potentially large delay caused by C_3 on C_2. In Figure 2d, C_3's incast receiver with heavy load would significantly delay C_2 on *out*.3, resulting in drastically inflated CCT for C_2 when compared with Figure 2c.

Hence, we have *Observation 2: Coflow placement should avoid contentions on endpoints with heavy traffic load.* When contentions among Coflows are unavoidable, the delay in the overall CCT caused by contentions is related to the amount of contenting traffic. This is because longer traffic queues at the contention points (e.g. at the edge links in our problem) usually result in longer traffic delay, and on the other hand, shorter queue shorter delay. For example, the worst solution shown in Figure 2d would create a much longer queue waiting at *out*.4 than any queue in the other solutions.

4 ALGORITHM DESIGN

To the best of our knowledge, no efficient algorithm is known to solve our problem optimally in polynomial time. Thus, we focus on designing good placement heuristic.

We observe placement schemes extended from flow level strategy are insufficient (§ 3.1) due to negligence of the inter-flow relationship, since flow level metrics (e.g. flow size) does not necessarily reflect on the key features at the Coflow level (e.g. bottleneck). To characterize the inter-flow relationship in a Coflow's demand D, we derive the aggregated traffic demand at the senders (receivers), which can be represented by vectors $L^s[.]$ for senders and $L^r[.]$ for receivers (Line 1 in Algorithm 1). Vector elements indicate the traffic load at an endpoint, and endpoints with heavier load are are more likely to become the bottleneck and delay CCT due to heavier contentions at the intra-Coflow level.

To characterize the inter-Coflow contentions, we calculate the traffic load on each input and output port by aggregating the remaining demand from all existing Coflows, which can be represented by vectors $E^s[.]$ for input ports and $E^r[.]$ for output ports. Under a set of active Coflows $\mathbf{C}$, we have $\forall in.i : E^s[in.i] = \sum_{C\in\mathbf{C}} \sum_j d'_{i,j}$, and $\forall out.j : E^r[out.j] = \sum_{C\in\mathbf{C}} \sum_i d'_{i,j}$, where $d'_{i,j} \in D'$ is a Coflow C's traffic demand from $in.i$ to $out.j$ ($d'_{i,j} = 0$ if demand is fulfilled or never requested).

$E^s[.]$ and $E^r[.]$ are generic measurements which consider all active Coflows in the network, and they are independent of the underlying network scheduler. $E^s[.]$ and $E^r[.]$ offer an estimation of the length of traffic queues behind each port. They are useful to indicate the overall traffic delay associated with a port, because ports with higher values are more likely to have longer traffic queues, which leads to larger average delay, even though the network scheduler may try to orchestrate resource allocation among Coflows to trade off their delays. Besides, these measurements are more practical because they do not involve bookkeeping the states for each individual flow as required in prior work for flow placement [14].

By leveraging these measurements that characterize a Coflow's traffic demand and the underlying network states, we design our Coflow placement algorithm, **2D-Placement**, as shown in Algorithm 1. 2D-Placement consists of two steps. First, 2D-Placement calculates the traffic demand requested on each endpoint for the Coflow to place, i.e. $L^s[.]$ and $L^r[.]$, as shown in Line 1. Second, 2D-Placement considers each sender (or receiver) in the descending order of their requested demand, and place the sender (or receiver) onto the input (or output) port

Algorithm 1 2D-Placement

```
Input: Coflow to place $C_{new}$, remaining load $E^s[.]$ and $E^r[.]$
Output: Placement of all senders $P^s(.)$ and receivers $P^r(.)$
1: for all $(s_i, r_j, d_{i,j})$ in $C_{new}$ do    ▷ Requested load
2:     Load on sender $L^s[s_i]$ += $d_{i,j}$
3:     Load on receiver $L^r[r_j]$ += $d_{i,j}$
4: end for
5: for all $s_i$ in the descending order of $L^s[.]$ do
6:     $P^s(s_i) = \text{argmin}\, E^s[.]$    ▷ Place sender
7:     $E^s[P^s(s_i)]$ += $L^s[s_i]$    ▷ Update load on port
8: end for
9: for all $r_j$ in the descending order of $L^r[.]$ do
10:     $P^r(r_j) = \text{argmin}\, E^r[.]$    ▷ Place receiver
11:     $E^r[P^r(r_j)]$ += $L^r[r_j]$
12: end for
```

with the minimum traffic load as described by $E^s[.]$ (or $E^r[.]$), as shown in Line 6 (or Line 10).

Our algorithm simultaneously achieves the two goals desired for Coflow placement. First, by sorting endpoints' traffic load, we prioritize endpoints that are more likely to become the bottleneck due to heavier intra-Coflow contentions, and place these critical endpoints on a port with more resource. Second, by finding a port with less traffic load, we reduce the amount of contending traffic across Coflows and thus the delay of queue waiting.

Examples Our algorithm obtains the optimal solutions for both examples shown in Figure 1 and Figure 2. In Figure 1, we start by placing heavy sender s_3 with more demand onto the least loaded input port $in.1$ and heavy receiver r_1 onto $out.4$, yielding the optimal solution as shown in the left most column of Figure 1b. In Figure 2, our algorithm will select the least loaded input ports $in.4$, $in.2$, $in.3$ for senders and the least loaded output port $out.1$ for the only receiver of C_3, yielding the optimal solution in Figure 2b.

Complexity To record network states, our algorithm only requires two vectors of size N, i.e. $E^s[.]$ and $E^r[.]$, where N is the number of ports. To place a Coflow with n_s senders, and n_r receivers, our algorithm need to calculate and sort $L^s[.]$ and $L^r[.]$, while $E^s[.]$ and $E^r[.]$ can be updated dynamically and computed asynchronously for fast look up of less loaded ports. Thus, our algorithm's time complexity is $O(n^2)$ where $n = \max(n_s, n_r)$.

5 EVALUATIONS

5.1 Methodology

Simulator: We implement a flow-level simulator [2] with various algorithms for Coflow placement and network scheduling. We implement two network schedulers for Coflows, i.e. Varys [6] and Aalo [5]. Both schedulers are designed to optimize the average CCT for a set of Coflows with *predetermined placement*, by prioritizing Coflows that are expected to finish faster with smaller bottleneck of their network demand $L(D')$. Varys assumes accurate Coflow traffic request, and Aalo tries to approximate Varys with unknown Coflow sizes so as to tolerate error in the requested demand.

Workload: We use a realistic workload based on a one-hour MapReduce trace collected from a Facebook production cluster [1]. The trace contains more than 500 Coflows observed in a 150-port fabric with exact inter-arrival times. The Coflows are in various structures (one-to-one, one-to-many, many-to-one and many-to-many).

Baseline: We compare against *Neat* [14], a state-of-the-art flow placement algorithm optimized with the awareness of network scheduling policy. To place a Coflow, Neat sequentially considers each constituent flow in their descending order of byte sizes, so as to avoid straggler flows and reduce the average CCT. Upon placing each flow, Neat opportunistically selects a node that minimizes the *estimated* sum of flow completion time (FCT) for the endpoint whose location is undetermined. To estimate the sum of FCTs for a flow's placement, Neat requires all flow and Coflow states, as well as the traffic priority under a specific network scheduling policy. In our evaluation, Neat uses the same traffic priority adopted by the network schedulers, i.e. prioritizing small Coflows that are expected to finish faster.

To evaluate Coflow placement with CCTs, a Coflow's requested demand should trigger actual traffic in the network. To do so, we require all placement algorithms to select distinct nodes for all senders and receivers for a Coflow whose $n_s + n_r \leq N$, where N is the number of nodes, and n_s (n_r) is the number of senders (receivers) in the Coflow. For a Coflow whose $n_s + n_r > N$, we allow reusing nodes between senders and receivers, but senders (or receivers) should reside on distinct nodes.

5.2 Coflow Performance Improvement

Improvement in average CCT: We begin our evaluation by comparing the average CCT under various scenarios involving different network schedulers and a range of traffic load. To evaluate performance under different traffic load, we scale up (down) the Coflow's traffic size with a factor, while preserving Coflows' structures and arrival time. Scale factor less than 1 means lighter traffic load than the default setting of the trace.

Table 1 shows 2D-Placement improves the average CCT by up to 26%. When scale factor > 1.5, CCTs inflate significantly due to overly high load. When scale factor is between 0.25 and 0.5, 2D-Placement remains the better placement scheme with normalized average CCT ≤ 0.98. Under extremely light load with scale factor < 0.25, 2D-Placement and Neat are close in performance,

Scale factor	$\times0.5$	$\times0.75$	$\times1$	$\times1.25$	$\times1.5$
Aalo	0.87	0.82	0.77	0.77	0.87
Varys	1.00	0.96	0.79	0.74	0.78

Table 1: 2D-Placement's average CCT normalized on Neat's average CCT. Normalized average CCT less than 1 means 2D-Placement is better.

since the available bandwidth for a Coflow is generally sufficient across the entire network, and therefore Coflow performance is less sensitive to placement and mainly depends on its demand bottleneck. This agrees with our previous observation (§ 3.1). Under medium or heavy workload that would result in heterogeneous network load and thus the available bandwidth for a Coflow is not uniformly distributed, Coflow placement is critical to improve Coflow performance.

Improvement in individual CCT: To compare two placement schemes more closely, we examine the individual CCTs under the default setting without traffic scaling. We begin with the metric used in previous Coflow studies [5, 6, 16], which is the CCT ratio between competing schemes. We find 2D-Placement is 0.995× (0.998×) of Neat on average under Aalo (Varys) scheduling. On a per Coflow basis, 2D-Placement only slightly improves over Neat.

At first glance, this seems to disagree with our previous results, which show much more drastic improvement in the average CCT. However, it is not hard to explain the difference. Prioritized Coflows are indifferent to placement, because the network scheduler usually allows these traffic to preempt bandwidth resource and thus their CCTs mainly depends on their demand bottleneck. Under both Varys and Aalo, smaller Coflows are prioritized. Since there are many more small Coflows in the trace, the pairwise comparison does not make much difference.

On the other hand, large Coflows are more sensitive to Coflow placement for several reasons. First, when they are less prioritized by the underlying network scheduler, they are more likely to be delayed due to waiting after or sharing bandwidth with other Coflows. Second, due to large traffic size, large Coflows usually stay in the network for longer time, and thus they are more likely to experience interruptions from new arrivals of higher or equal priority traffic. In our evaluation, large Coflows benefit from 2D-Placement, resulting in a drastic improvement in the average CCT.

We confirm our observations with Figure 3. We consider a Coflow to be *large* if the ratio of its demand bottleneck over link bandwidth $L(D)/B \geq 60$ (i.e. the Coflow takes at least 60 seconds to finish) under the original setting of the trace. Large Coflows account for 4.2% of all Coflows and 95.6% of all bytes. All large Coflows are many-to-many Coflows with up to 147 senders and 145 receivers.

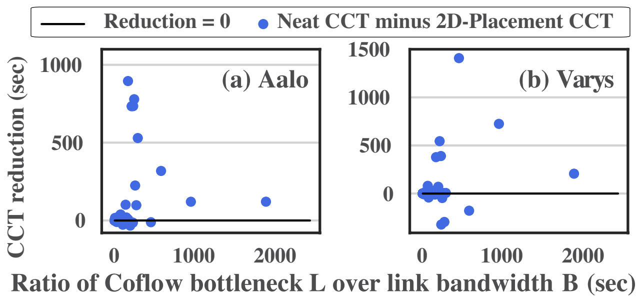

Figure 3: Individual CCT reduction of 2D-Placement from Neat. (a) Aalo. (b) Varys. Coflows are spread along the x-axis on the bottleneck L of their traffic demand D. 2D-Placement is better with larger values.

For small Coflows on the left of x-axis in Figure 3, their CCT are not much different under 2D-Placement or Neat, because their traffic is prioritized. For large Coflows where most of the bytes come from, we observe Neat performs worse on average. This is because Neat's estimated sum of FCTs does not necessarily translate to the sum of CCTs as in our objective. Neat's estimation is based on a simplified assumption: for Coflows that share a link, each Coflow will assign *proportionally equal* traffic (with respect to each Coflow's total traffic size) on the link that the Coflows are contending for. However, such assumption usually does not hold true in practice because Coflows have varying structures and do not split traffic in the same way. As a result, Neat may make poor placement decisions for Coflows due to its overly simplified assumption. Besides, Neat's estimation is prone to error when the runtime traffic priority deviates from the priority as assumed by Neat (see below "Sensitivity to network scheduling").

In contrast, 2D-Placement achieves better CCTs for large Coflows: 2D-Placement is only 0.85× (0.92×) of Neat on average under Aalo (Varys) scheduling. A large Coflow is hard to place because it usually has more senders, more receivers and larger traffic volume, and yet a large Coflow is also more sensitive to placement which has long term impact on the Coflow's performance. 2D-Placement allows large Coflows to finish faster, which translates to a more favorable average CCT.

Sensitivity to network scheduling: We compare how one placement scheme performs under different network schedulers with the default traffic settings. Note that even under the same settings, Aalo is usually worse than Varys because it may temporarily allow Coflows to violate the smallest-Coflow-first priority due to lack of precise knowledge of Coflow sizes. Nevertheless, for small Coflows that are prioritized under both schedulers, neither placement scheme makes much difference. Thus, for small Coflows, Aalo is 1.27× (1.30×) of Varys on average under 2D-Placement (under Neat). However, for large Coflows that are more sensitive to placement, 2D-Placement allows Aalo to perform much closer to Varys. Under 2D-Placement, Aalo is only 1.65× of Varys on average for large Coflows. Under Neat, however, Aalo becomes much worse with 1.88× of Varys on average for large Coflows. Neat optimizes placement based on a specific traffic priority used in the network scheduler. However, when the network scheduler does not fully observe the priority during runtime, Neat may make poor placement decisions due to estimation errors. For example, a high priority Coflow assumed by Neat could be delayed in practice, which prolongs the average CCT. In contrast, the load and demand metrics used 2D-Placement are generic measurements that optimize the CCTs by reducing contentions. 2D-Placement is independent on the underlying network scheduler and thus is more tolerable to dynamics in network scheduling during runtime.

6 RELATED WORK

To the best of our knowledge, this work is the first to study Coflow placement problem with considerations of the inter-flow relationship in Coflows. A range of recent proposals [5, 6, 10, 16] demonstrate improved application-level communication efficiency with Coflow aware network scheduling under predetermined Coflow placement. Coflow placement problem is different from task placement [7, 11, 15], which is more interested in individual task performance or the cluster wide metrics such as resource utilization. In contrast, the performance of a Coflow must consider all constituent flows. As a result, flow placement strategies [14] may yield suboptimal solution due to negligence of the inter-flow relationship in a Coflow. Different from designs [12] that speed up Coflow completion by colocating senders and receivers to reduce Coflow traffic demand, our work optimizes placement in a general scenario where colocation is not feasible, e.g. in the case where storage nodes do not host computation tasks from data processing jobs.

7 CONCLUSION

Coflow placement, despite its decisive impact on Coflows' performance, is a problem long been overlooked. This paper presents the first study on the Coflow placement problem with careful considerations of the inter-flow relationship in Coflows. Avoiding bottleneck delay and reducing bottleneck contention are the keys to good placement for Coflows. We design a Coflow placement algorithm that simultaneously achieves these desirable goals. Evaluations show our algorithm improves the average CCT by up to 26%. We demonstrate the benefits of exploiting inter-flow relationship in finding good placements for Coflows, which enables the underlying network scheduler to achieve better Coflow performance.

ACKNOWLEDGMENTS

We thank the BOLD Lab members and the anonymous reviewers for their useful feedback. We also thank Mosharaf Chowdhury and Ali Munir for their comments and for sharing traffic traces and simulators. This research is sponsored by the NSF under CNS-1422925 and CNS-1305379.

REFERENCES

[1] 2014. Coflow Benchmark Based on Facebook Traces. (2014). https://github.com/coflow/coflow-benchmark.
[2] 2017. 2D-Placement simulator. (2017). https://github.com/sunnyxhuang/2D-Placement.
[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *SIGCOMM '08*.
[4] Mosharaf Chowdhury and Ion Stoica. 2012. Coflow: A networking abstraction for cluster applications. In *HotNets '12*.
[5] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient Coflow Scheduling without Prior Knowledge. In *SIGCOMM '15*.
[6] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient coflow scheduling with Varys. In *SIGCOMM '14*.
[7] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. In *SIGCOMM '14*.
[8] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. 2016. Altruistic Scheduling in Multi-Resource Clusters. In *OSDI '16*.
[9] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A scalable and flexible data center network. In *SIGCOMM '09*.
[10] Xin Sunny Huang, Xiaoye Steven Sun, and T. S. Eugene Ng. 2016. Sunflow: Efficient Optical Circuit Scheduling for Coflows. In *CoNEXT '16*.
[11] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. 2009. Quincy: fair scheduling for distributed computing clusters. In *SOSP '09*.
[12] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. 2015. Network-Aware Scheduling for Data-Parallel Jobs: Plan When You Can. In *SIGCOMM '15*.
[13] Shouxi Luo, Hongfang Yu, Yangming Zhao, Sheng Wang, Shui Yu, and Lemin Li. 2016. Towards Practical and Near-optimal Coflow Scheduling for Data Center Networks. In *IEEE TPDS '16*.
[14] Ali Munir, Ting He, Ramya Raghavendra, Franck Le, and Alex X. Liu. 2016. Network Scheduling Aware Task Placement in Datacenters. In *CoNEXT '16*.
[15] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *EuroSys '15*.
[16] Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, and Yanhui Geng. 2016. CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark. In *SIGCOMM '16*.