Weaver: Efficient Coflow Scheduling in Heterogeneous Parallel Networks

Xin Sunny Huang*, Yiting Xia†, T. S. Eugene Ng*

Rice University*, Facebook, Inc.†

***Abstract*—Leveraging application-level requirements expressed in Coflows has been shown to improve application-level communication efficiency. However, most existing works assume *all* application traffic is serviced by *one* monolithic network. This over-simplified assumption is no longer sufficient in a modern, evolving data center which operates on *multiple* generations of network fabrics, an architecture that we define as *Heterogeneous Parallel Networks* (HPNs). In this paper, we present the first scheduler, called *Weaver*, that addresses the Coflow management problem in HPNs.**

To exploit HPNs fully, achieving high communication efficiency for applications is crucial, yet it is also challenging because of the complex traffic patterns in applications *and* the heterogeneous bandwidth distribution in HPNs. Weaver addresses these challenges at two levels. At the microscopic level, for each application, Weaver leverages an efficient algorithm to exploit the distributed bandwidth in HPNs, which we proved to be within a constant factor of the optimal. At the macroscopic level involving multiple applications, Weaver can adopt a range of application traffic scheduling policies as desired by the system operator. Under realistic traffic, Weaver enables HPNs to service Coflows as efficiently as a monolithic network with equivalent aggregated capacity.

I. Introduction

A range of recent works [1, 2, 3, 4, 5] have demonstrated the benefits of application-aware network scheduling by exploiting the structured traffic pattern from distributed applications. Distributed applications in data centers often communicate with a collection of related flows. Prevalent examples are the parallel flows involved in executing a query in distributed databases [6], a read or write in distributed storage systems [7], and the data shuffle phase in distributed MapReduce jobs [8, 9], etc. This typical communication pattern commonly seen in data centers is captured concisely by the Coflow traffic abstraction (Section II). A Coflow represents a collection of related flows whose performance goal is to finish all related flows as soon as possible. Coflow-aware scheduling benefits distributed applications by avoiding stragglers [1, 2, 4] and improving resource utilizations [10].

Prior works on Coflow scheduling mainly focus on the *single-core* model, a network model that has been widely used not only for Coflow studies [1, 2, 3, 4, 5], but also for other works on data center network [11]. The single-core model abstracts the whole network fabric as *one* non-blocking N-port switch, an assumption considered practical because topological designs such as Fat-tree or Clos [12, 13] enable building a data center network with full bisection bandwidth. Nevertheless, this over-simplified model is limited to the scope where *all* Coflow workload is exclusively supported by *one* generation of network fabric.

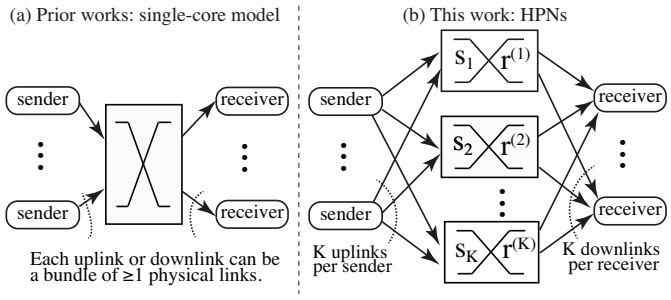

Fig. 1: Comparison of network models. (a) Prior works: single-core model. (b) This work: Heterogeneous Parallel Networks (HPNs).

However, this single-core model is no longer sufficient under recent technology trends [14, 15, 16, 17, 18]. We observe that an evolving data center would operate on *multiple generations of networks in parallel* [19], so as to exploit the shrinking gap in network speed and to derive benefits for research and development endeavors (Section II). We define such network settings as *heterogeneous parallel networks* (HPNs), an architecture that is based on multiple network cores operating in parallel (Figure 1). A *network core* represents one generation of network fabric that can service all hosts in a cluster. Multiple network cores make up *parallel networks* to provide a large amount of aggregated bandwidth by servicing traffic simultaneously. HPNs evolve over time by adding new cores to work in parallel with the existing ones, and by retiring old cores whose capacity is significantly less than the youngest core to reduce system complexity. At any time, cores with various transmission capacities coexist. In HPNs, the conventional single-core network model is insufficient.

Based on these observations, we ask a timely and important question: how to best service Coflows in the data center supported by multiple generations of network fabrics, i.e. the HPNs? Coflow scheduling is known to be a hard problem (NP-hard, [1]) even in a single-core network, because one application often depends on multiple flows among its distributed compute nodes, and thus resource management solutions at the individual flow level [20, 21] often fail to translate benefits to the Coflow level [1]. The Coflow scheduling problem becomes even more challenging in HPNs because bandwidth resources are *distributed* among *heterogeneous* network cores.

†Work done while Yiting Xia was at Rice University.

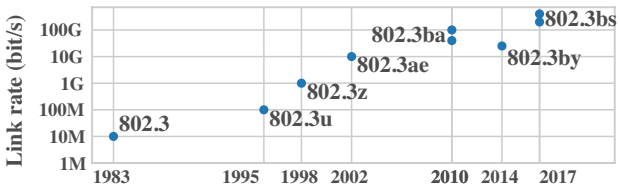

Fig. 2: Link rate and the year first introduced in IEEE 802.3

Thus, resource management also needs to decide how Coflow traffic should be assigned to the appropriate core to utilize the core's bandwidth. Furthermore, achieving high-efficiency for Coflow scheduling usually requires global coordinations, but a network core in HPNs should operate as independently as possible for several practical considerations (Section III).

In this paper, we present *Weaver*, the first Coflow scheduler for a modern data center operating on HPNs. To address the above technical challenges, our key design choice is to decouple (1) bandwidth allocation (BA) within each core and (2) traffic-to-core assignment (TA). For BA, Weaver designates a component for each core to coordinate the bandwidth allocation for the traffic within the core. This design not only decouples a core's traffic control from other cores', but also allows Weaver to accommodate a variety of traffic priority policies to achieve the desired scheduling objective. For TA, Weaver incorporates a flow-level traffic assignment component to maximally exploit the distributed bandwidth among heterogeneous cores. While traffic assignment optimizing for Coflow performance is a hard problem (NP-hard, Section IV-A), we propose a TA algorithm that is provably within a constant factor of the optimal. With testbed implementations and large-scale simulations, we demonstrate experimentally that Weaver enables HPNs to service Coflows as efficiently as a monolithic network with equivalent aggregated capacity.

II. Background

Heterogeneous parallel networks (HPNs): HPNs are deployed in data centers today for several compelling advantages, such as (1) exploiting the shrinking generation gap in network speed, (2) serving the transitional period during network upgrade to provide alternative links, and (3) enabling research with new experimental technology in situ. We discuss these important use cases of HPNs as follows.

In recent years, the growth of traffic demand in data centers is winning the race against network link capacity. Google recently reported its aggregated server traffic increases by 50× in only 6 years from 2008 to 2014 [19]. Similar anecdotes are often heard [14, 22, 23]. Such immense growth is driven by bandwidth hungry distributed applications, such as distributed storage and large-scale data analytics. In contrast, the marginal gain in speed for new generations of link technology is *shrinking*. Figure 2 shows the year when various link rates were introduced in IEEE 802.3, the industrial standard for wired Ethernet. Before 2002, each new generation used to deliver 10× improvement over the previous one. However, this 10× boom has disappeared after 10 Gbps. Nowadays, when 100 Gbps links are prohibitively expensive,[1] 25 Gbps or 40 Gbps remain more attractive to data center operators [18, 19, 25], i.e. a mere 2.5× or 4× improvement over 10 Gbps. The shrinking gap is due to many reasons, the most important being the technological barrier: new technology to support high link rates at low cost is yet to be developed [15, 16, 17].

TABLE I: Notations

Symbol	Definition
K	number of parallel network cores
N	port count of parallel networks
$in.i$, $out.j$	i-th input port, j-th output port
$r^{(k)}$	link capacity of the k-th core
$\|C\|$	Number of flows in a Coflow C
$\mathbf{D}^{(k)}$	a Coflow C's traffic demand on the k-th core
$L(\mathbf{D})$	bottleneck of matrix $\mathbf{D}$ (Section IV-B)
T_o^f or T_o^s	CCT lower bound if flow splitting is allowed (Table II) or prohibited (Table III), respectively

These conflicting stances call for an incremental approach for network upgrades. On one hand, traffic growth presses for more bandwidth [18, 19]. On the other, building a brand new network on 10× improved link rate becomes prohibitively expensive [18]. So operators turn to the best affordable link rate, usually 2.5× or 4× of the legacy network [18, 26]. The result is that, relative to a new network, a legacy network can still service a considerable amount of traffic. For example, there is significant incentive to retain a legacy 10 Gbps network because it could service 20% of traffic together with a new 40 Gbps network of the same topology when both work in parallel.

Operating multiple generations of networks in parallel also provides additional benefits to facilitate research and development endeavors. For instance, to deploy an unproven new network, Google retains the legacy network to assure quality of service [19], so that the legacy network may handle important traffic until the new network has been rigorously tested over time. In addition, parallel networks can also be added to test new experimental technology in situ without incurring host downtime, a common practice widely adopted by the industry [19] as well as research institutes [27].

Based on these observations, a case could be made for operating two, or perhaps even three generations of networks in parallel that provide the above technical advantages. We have defined such network settings as *heterogeneous parallel networks* (HPNs) in Section I. As shown in Figure 1, without loss of generality, we abstract the HPNs as K non-blocking N-port switches $\{s_1, ..., s_K\}$, each representing one network core with link rate $\{r^{(1)}, ..., r^{(K)}\}$ respectively. This abstract model is simple yet practical because topology designs such as Fat-tree or Clos [12] enable building a network with full bisection bandwidth. Among the N sender (or receiver) nodes, the i-th sender (or the j-th receiver) is connected to the i-th input (or the j-th output) port of each parallel switch. Hence,

[1] Price of one transceiver to support 1km cable distance [24]: 10 Gbps at $34, 40 Gbps at $220, 100 Gbps at $700.

TABLE II: Generic Assignment Problem

	Formula	Definition
Variables:	$\forall k : d_{i,j}^{(k)} \in \mathbf{D}^{(k)}$	demand sent by k-th core
Goal:	$\min T$	minimize CCT
Constraints:		
1)	$\forall i,j : \sum_{k=1}^{K} d_{i,j}^{(k)} = d_{i,j}$	demand satisfaction
2)	$\forall i,k : \sum_{j=1}^{N} d_{i,j}^{(k)} \leq r^{(k)}T$	k-th core's inbound cap
3)	$\forall j,k : \sum_{i=1}^{N} d_{i,j}^{(k)} \leq r^{(k)}T$	k-th core's outbound cap

TABLE III: Flow-level Assignment Problem

	Formula	Definition
Variables:	$\forall k : I_{i,j}^{(k)} \in \{1,0\}$	$f_{i,j}$ sent by k-th core if $I_{i,j}^{(k)}$=1.
Goal:	$\min T$	minimize CCT
Constraints:		
1)	$\forall f_{i,j} \in \mathbf{D} : \sum_{k=1}^{K} I_{i,j}^{(k)} = 1$	each flow sent by one core
2)	$\forall i,k : \sum_{j=1}^{N} d_{i,j} I_{i,j}^{(k)} \leq r^{(k)}T$	k-th core's inbound cap
3)	$\forall j,k : \sum_{i=1}^{N} d_{i,j} I_{i,j}^{(k)} \leq r^{(k)}T$	k-th core's outbound cap

each sender (or receiver) simultaneously has K uplinks (or downlinks). Each uplink (or downlink) can be a bundle of multiple physical links in the actual topology. In practice, the sender (or receiver) nodes are most likely Top-of-Rack (ToR) switches each connected to a group of hosts. Note that a network core in our HPNs is an autonomous entity that transmits data between endpoints without interference from other cores. It is different from a core switch in a Clos network, where core switches are merely interchangeable, intermediate nodes to forward traffic redistributed arbitrarily by upstream switches in the network.

Coflow traffic abstraction: Data-parallel distributed computation frameworks [6, 7, 8, 9] usually organize their pipeline in stages, and each stage need data transfer on a collection of related flows among two groups of machines. Coflow is a traffic abstraction designed to concisely characterize this typical communication pattern. A Coflow is defined by the endpoints and byte size of each flow in the Coflow. A Coflow can represent any communication structure, such as many-to-many, one-to-many, and many-to-one, etc. The traffic demand of a Coflow can be represented in a demand matrix $\mathbf{D}$, where each element $d_{i,j} \in \mathbf{D}$ indicates flow $f_{i,j}$ transfers $d_{i,j}$ amount of data from $in.i$ to $out.j$.

The Coflow transfer phase has paramount impact on the application performance because the application usually relies on the completion of a Coflow to continue computation or generate final results [6, 8, 9] and to fulfill data management goals [7]. Thus, a Coflow's performance is measured by its Coflow Completion Time (CCT), which is the duration to finish all constituent flows in the Coflow. For a Coflow C, denote its arrival time as t^{Arr}, and the finish time of the flow $f_{i,j} \in C$ as $t_{i,j}^{F}$. CCT is defined as $\max_{\forall f_{i,j} \in C}(t_{i,j}^{F} - t^{\text{Arr}})$. At the intra-Coflow level, the goal is to minimize CCT, so as to speed up application level performance. This scheduling objective is commonly used in all prior studies [1, 2, 3, 4, 5].

III. Weaver Overview

This section presents an overview of Weaver, the first Coflow scheduler for HPNs. Similar to the scheduler designs in prior works [1, 2, 3, 4], Weaver serves as a logically centralized coordinator for Coflow transmissions in HPNs. Weaver accepts traffic requests expressed in Coflows from applications. Given multiple Coflow requests, Weaver must decide how Coflow traffic should be assigned to the appropriate core and, within each core, how to allocate the bandwidth for constituent flows in Coflows.

Under the single-core assumption, prior Coflow-based solutions generally apply global control on traffic priority and bandwidth allocation [1, 2, 3, 4, 5], as well as routing [28], for all constituent flows in Coflows to achieve high communication efficiency. In HPNs, however, global control is challenging. The heterogeneous cores may differ in various levels of their network stack, such as control software, transmission protocol, and hardware generation, etc. In certain cases, it might not be practical to jointly schedule Coflows across different cores. For example, a network core based on a new hardware stack would require its specialized bandwidth allocation policy [4], and it is unknown how different types of bandwidth allocation policy, such as [4] and [1], may incorporate efficiently with each other. Thus, a network core should operate as independently as possible.

Under these technical considerations, Weaver decouples bandwidth allocation within each core and traffic-to-core assignment, so that a core may deploy its own bandwidth allocation policy for Coflows and a Coflow may still exploit bandwidth distributed in multiple cores. This approach also enables Weaver to accommodate a variety of Coflow scheduling policies desired by the system operator (Section V). As an overview, Weaver consists of the following components.

Traffic Assignment (TA): A new Coflow request first goes to TA, which assigns the requested demand to be fulfilled by each BA (discussed below). For a Coflow C with traffic demand $\mathbf{D}$, its traffic assigned to the K BAs can be represented in demand matrices $\{\mathbf{D}^{(1)}, ..., \mathbf{D}^{(K)}\}$, where each element $d_{i,j}^{(k)} \in \mathbf{D}^{(k)}$ indicates the amount of data in flow $f_{i,j}$ from Coflow C to be fulfilled by the BA on the k-th core.

Bandwidth Allocation (BA): Weaver designates a BA component for each core to allocate bandwidth for traffic within the core. All BAs periodically report traffic status in the system to a Status Server (SS), which in return helps BAs and TA to optimize scheduling decisions at the inter-Coflow level (Section V). Denote $r_{i,j}^{(k)}$ as the bandwidth allocated for the link from $in.i$ to $out.j$ in the k-th core. BAs ensure bandwidth constraints are satisfied with $\forall k,j : \sum_{i=1}^{N} r_{i,j}^{(k)} \leq r^{(k)}$ and $\forall k,i : \sum_{j=1}^{N} r_{i,j}^{(k)} \leq r^{(k)}$.

IV. Traffic Assignment

When a new Coflow arrives, TA assigns its requested demand to network cores. In this section, we study how to assign a Coflow's demand at the intra-Coflow level.

A. Problem Formulation and Hardness

We start by formulating the traffic assignment problem. TA decomposes C into K children Coflows, i.e. one child per BA. Following the common defination of CCT (Section II), the CCT for the parent C is the maximum of the CCTs of all its children, i.e. $CCT = \max_{\forall k}(CCT^{(k)})$.

At first glance, the traffic assignment problem of a parent Coflow can be formulated as in Table II. As we will show in Section IV-B, this problem can be solved optimally by setting $d_{i,j}^{(k)}$ proportionally to $r^{(k)}$, i.e. $d_{i,j}^{(k)} = d_{i,j}\frac{r^{(k)}}{R}$, where R=$\sum_{k=1}^{K} r^{(k)}$. However, this requires *flow splitting*, i.e. splitting a flow into multiple subflows to transmit through different network cores. Flow splitting is less practical due to packet reordering. Therefore, we focus on a solution that assigns Coflow traffic at the flow level, so that data in a flow is assigned to the same core, while different flows may transmit through different cores. We formulate this problem in Table III.

This problem (Table III) is NP-hard [29, 30, 31] under a simplified case of one-to-many (or many-to-one) Coflow, where we only need to consider the capacity constraints of the K input (or output) ports associated with the K cores in HPNs from the same sender (or to the same receiver). Under these cases, our problem is equivalent to scheduling independent tasks on heterogeneous processors to minimize the finish time, where $|C|$ independent flows (tasks) in a Coflow C are to be assigned on K network cores of various link rates (processors with various service speed) without flow splitting (task preemption) to minimize CCT (the finish time of the last task). However, our problem is more general because Coflows have various structures other than one-to-many and many-to-one, which requires us to *jointly consider the inbound and outbound capacity for all input and output ports of all cores*. Classic heuristics [29, 30, 31] designed for heterogeneous processors assume one resource type on a processor. Therefore, they cannot be applied to our problem which involves $2N$ types of resources (bandwidth of N input ports and N output ports) on each processor (network core). The search space for each Coflow C is exponentially large – there are $K^{|C|}$ possible solutions to assign the $|C|$ flows from the Coflow C to K cores, and $|C|$ can be as large as N^2. For example, in a widely used Coflow benchmark [32], $> 30\%$ Coflows each has > 30 flows, which translates to > 1 billion possible assignments to be considered in an exhaustive search for a Coflow under the most common case of $K = 2$.

B. CCT Lower Bounds

To facilitate theoretical analysis, we begin by deriving the CCT lower bounds for our problems in Table II and Table III. Denote T_o^f as the optimal CCT without flow splitting, and T_o^s as the optimal CCT allowing flow splitting. The problem without flow splitting is a special case of the problem allowing flow splitting, and thus T_o^f is achievable under the constraints of T_o^s. Therefore, we have $T_o^s \leq T_o^f$.

We then introduce the *bottleneck* metric to characterize a Coflow's traffic demand matrix $\mathbf{D}$. We define its bottleneck as the maximum of column sums and row sums in the matrix, i.e. $L(\mathbf{D})$ = $\max\left(\max_{\forall i}\sum_{j=1}^{N} d_{i,j}, \max_{\forall j}\sum_{i=1}^{N} d_{i,j}\right)$.

Algorithm 1 Weaver's Flow-level TA Algorithm

```
1: procedure ASSIGN(Coflow C, link rates r^(1),..., r^(K))
2:     ∀k : D^(k) = ∅        ▷ Demand assigned to the k-th network
3:     for all (i, j, d_{i,j}) in SORTED(C) do
                  ▷ Sort flows in the descending order of byte size
4:         if ∃k : L(D^(k) ∪ d_{i,j}) > L(D^(k)) then
                  ▷ Critical flow: pick a network to minimize CCT
5:             k_min = argmin_k L(D^(k) ∪ d_{i,j}) / r^(k)
6:         else
                  ▷ Non-critical flow: pick a network to balance load
7:             k_min = argmin_k max(V1(D^(k) ∪ d_{i,j})[i], V2(D^(k) ∪ d_{i,j})[j]) / r^(k)
8:         end if
9:         D^(k_min) ← D^(k_min) ∪ d_{i,j},   I_{i,j}^(k_min) = 1
                                              ▷ Update assignment
10:    end for
11:    return {I^(1), ..., I^(K)}
12: end procedure
```

The CCT lower bound for a child Coflow $C^{(k)}$ on the k-th core with link rate $r^{(k)}$ is $\frac{L(\mathbf{D}^{(k)})}{r^{(k)}}$[4]. Hence, the CCT of the parent Coflow is $\max_{\forall k}(\frac{L(\mathbf{D}^{(k)})}{r^{(k)}})$. The following theorem establishes that the CCT in HPNs is no less than the CCT lower bound in a monolithic network with equivalent aggregated link capacity $R = \sum_{k=1}^{K} r^{(k)}$.

Theorem 4.1: $\frac{L(\mathbf{D})}{R}$ is the CCT lower bound in a core with link rate $R = \sum_{k=1}^{K} r^{(k)}$, and $\frac{L(\mathbf{D}^{(k)})}{r^{(k)}}$ is the CCT lower bound in the k-th core with link rate $r^{(k)}$. We have $\max_{\forall k}(\frac{L(\mathbf{D}^{(k)})}{r^{(k)}}) \geq \frac{L(\mathbf{D})}{R}$.

Proof: See Appendix ■

Theorem 4.1 implies that an optimal solution for the problem that allows flow splitting is to set $d_{i,j}^{(k)}$ proportional to $r^{(k)}$, i.e. $d_{i,j}^{(k)}$=$\frac{r^{(k)}}{R}$, so that the CCT lower bound is achieved in each core, i.e. $CCT = CCT^{(1)} = ... = CCT^{(K)} = \frac{L(\mathbf{D})}{R}$. Thus $T_o^s = \frac{L(\mathbf{D})}{R}$.

C. Algorithm Design

Algorithm 1 presents Weaver's *flow-level* traffic assignment algorithm. We have shown in Section IV-A that our problem without flow splitting is NP-hard. We prove Weaver's TA algorithm is tightly bounded within a constant factor of T_o^s, the optimal of the relaxed problem that requires flow splitting. This optimality guarantee applies in the general case with no constraints on Coflow structures or HPNs configurations (Section IV-D). In other words, *without* flow splitting, Weaver's TA algorithm approximates well the performance of the stringent case that requires flow splitting. Towards this goal, we leverage three crucial techniques as follows.

Achieving optimality guarantee with careful assignment of critical flows. Weaver's TA algorithm considers flows one-by-one. The algorithm achieves its performance guarantee by classifying *critical* flows in a Coflow and assigning them to the network core that minimizes CCT after adding the critical

flow. Specifically, when Weaver's TA considers each flow, the flow is matched with all K network cores. A flow is defined as *critical* if adding the flow would increase the bottleneck of at least one core's demand matrix (line 4). In other words, if the flow is assigned to one of the affected core(s), the CCT of the child Coflow assigned to that core will increase, which may further increase the CCT for the parent Coflow, because the parent's CCT is the maximum of its children CCTs, as discussed in Section IV-A. For our optimality guarantee to hold true (Section IV-D1), a critical flow is assigned to the core with the minimum CCT for its child Coflow including the newly added flow (line 5). In this way, Weaver considers the heterogeneous core capacity and Coflow structures simultaneously to optimize Coflow performance.

Achieving better load balancing with busy ratio. The assignment of non-critical flows can be arbitrary for our optimality guarantee to hold, because none of the children CCTs will increase. Nevertheless, it is crucial to balance load among network cores, so as to improve performance of inter-Coflow scheduling when multiple Coflows coexist in the system. We will discuss the details of inter-Coflow scheduling later in this paper (Section V). When multiple Coflows coexist in the system, given that the critical flow condition is observed to achieve the optimality guarantee, TA should strike a balanced load among various cores for a new Coflow, so as to reduce contentions between Coflows within one core. Otherwise, flows on an overloaded core may be heavily delayed, prolonging CCTs of the affected Coflows.

To balance load for non-critical flows, Weaver takes a "worst-fit" approach to match the flow with the least busy core. We avoid using the bottleneck of a core's demand matrix $\mathbf{D}^{(k)}$, because the bottleneck is biased by the highly loaded input (or output) port, which is usually not the port needed by the flow. To better evaluate the load of a core for the flow, we instead use the ratio of the maximum load of the flow's input and output port over the core's link capacity (line 7). The ratio considers a core's load on the relevant ports of the flow, as well as the heterogeneous link capacity. A smaller ratio indicates that the core is relatively less busy on the flow's path, so the flow is assigned to the core with the minimum "busy ratio".

Achieving better assignment with assignment ordering. The ordering of the flows to be considered for traffic assignment can be arbitrary (line 3) for our optimality guarantee to hold true. Nevertheless, we recommend prioritizing assignment for flows of larger size because they are more likely to finish later and impact the CCT. Therefore, before feeding a Coflow as input to the algorithm, flows in the Coflow may be optionally sorted in their descending order of flow sizes so that larger flows are considered earlier.

Example. Putting these crucial techniques together, Weaver's TA algorithm is shown as in Algorithm 1. We leverage an example in Figure 3 to demonstrate how Weaver assigns traffic for a Coflow C. In this example, we consider two cores, s_1 and s_2. s_1 has a lower link rate of $r^{(1)} = 1$, and s_2 has a higher link rate of $r^{(2)} = 4$.

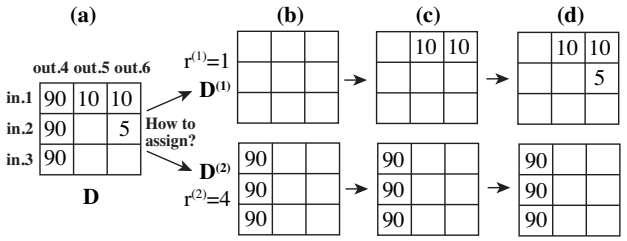

Fig. 3: Weaver's TA example: Assigning flows from a Coflow to two network cores, s_1 and s_2, with link capacity $r^{(1)} = 1$ and $r^{(2)} = 4$, respectively. (a) Coflow traffic demand. (b) Three critical flows are assigned to s_1 to minimize CCT. (c) Two critical flows are assigned to s_2 to minimize CCT. (d) One non-critical flows are assigned to s_2 to balance load and avoid overloading at $in.2$ on s_1.

$\mathbf{D}^{(k)}$ is Coflow C's traffic demand assigned to the k-th core. When considering a new Coflow for HPNs with K cores, all K matrices are initialized to be empty (line 2). When a flow is assigned to a specific core, the core's demand matrix is updated with the flow's byte size (line 9). In our example, we have $\mathbf{D}^{(1)}$ for s_1 and $\mathbf{D}^{(2)}$ for s_2.

As discussed in Section IV-C, *Weaver achieves better assignment by considering larger flows first.* Therefore, Weaver starts with the three flows to $out.4$, each flow of size 90.

Each of the flows with size 90 are critical, because adding the flow would increase the bottleneck of both $\mathbf{D}^{(1)}$ and $\mathbf{D}^{(2)}$ at $out.4$. To achieve its performance guarantee, *Weaver assigns a critical flow to the network core which minimizes CCT after adding the flow.* Therefore, all three flows, one flow at a time, are assigned to s_2 with higher link capacity $r^{(2)} = 4$, as shown in Figure 3b. Then, Weaver continues to consider the flows of size 10. Again, each of the them are critical due to increasing the bottleneck of $\mathbf{D}^{(2)}$ at $in.1$. Therefore, the flows are assigned to s_1 which yields lower $CCT = 2 \times 10/1 = 20$, when compared with s_2 and its $CCT = 3 \times 90/4 = 67.5$, as shown in Figure 3c.

In Figure 3d, the flow of size 5 is non-critical because adding the flow to either s_1 or s_2 would neither impact the bottleneck of $\mathbf{D}^{(1)}$ at $in.1$ nor the bottleneck of $\mathbf{D}^{(2)}$ at $out.4$. *Weaver uses the busy ratios to achieve load balancing.* The ratios on s_1 is $(10 + 5)/1 = 15$, lower than s_2's ratio of $(90 + 5)/4 = 23.75$. This indicates s_1 is less busy on the paths of the non-critical flow, so the flow is assigned to s_1.

D. Theoretical Analysis

1) Optimality Guarantee: Exploiting all cores in HPNs is crucial to optimize Coflow performance. A TA algorithm should never perform worse than operating the fastest core *alone*, otherwise it defeats the purpose of operating multiple cores. However, in the presence of slower cores, a Coflow might end up with worse performance than using the fastest core alone due to a deficient TA. For an extreme example, when the fastest core is mostly idle, CCT suffers if the Coflow is assigned to a slower core. In contrast, we prove that CCT under Algorithm 1 is never worse than assigning the whole Coflow to the fastest core, as shown in Theorem 4.2.

Theorem 4.2: Denote $\mathbf{D}^{(k)}$ as the matrix of traffic demand assigned for the k-th network core under Algorithm 1, and

$\sum_{k=1}^{K} \mathbf{D}^{(k)} = \mathbf{D}$, where $\mathbf{D}$ is the traffic demand of Coflow C, then $\max_{\forall k} \left(\frac{L(\mathbf{D}^{(k)})}{r^{(k)}} \right) \leq \min_{\forall k} (\frac{L(\mathbf{D})}{r^{(k)}})$.

Proof: See Appendix. ■

Theorem 4.1 and Theorem 4.2 imply that Algorithm 1 achieves a CCT within a constant factor (i.e. α in Theorem 4.3) of the optimal CCT under any specific configuration of HPNs, as shown in the following Theorem 4.3.

Theorem 4.3: Denote T as the CCT achieved by Algorithm 1, T_o^f as the optimal CCT without flow splitting, and T_o^s as the optimal CCT allowing flow splitting. Then we have $T \leq \alpha T_o^s \leq \alpha T_o^f$, where $\alpha = \frac{\sum_{k=1}^{K} r^{(k)}}{\max_{\forall k}(r^{(k)})} \leq K$.

Proof: See Appendix ■

In HPNs with K network cores, Theorem 4.3 shows that Algorithm 1's optimality approximation factor, or α, is always better than K in HPNs where parallel networks have heterogeneous link capacity. Under K uniform parallel networks, i.e. $r^{(1)} = ... = r^{(K)}$, $\alpha = K$. In the extreme case where *one* core provides the majority of bandwidth, $\alpha \to 1$.

Note that the optimality guarantee of Weaver's TA algorithm holds true regardless of the ordering of flows to be assigned or the assignment of non-critical flows. Nevertheless, special treatments for assignment ordering and assignment of non-critical flow are beneficial to achieve better Coflow performance in practice, as shown later in this paper (Section VI-C).

2) Time Complexity: For each Coflow C, Algorithm 1 scans each flow in C (line 3), which results in $|C|$ iterations. For each flow, K cores are compared to find the best that optimizes CCT after adding the flow (line 5 or line 7). Therefore, the time complexity of Algorithm 1 is $O(K|C|)$.

V. Bandwidth Allocation

In the presence of multiple Coflows, how to share bandwidth at the inter-Coflow level is also critical to optimize Coflow performance. At this inter-Coflow level, the scheduling objective will depend on the resource management policy employed by the system. A commonly studied objective is to minimize the average CCT, which is already NP-hard [1] in the simplified case of monolithic network (K=1). To achieve this objective, a variety of Coflow schedulers, such as Varys [1] and Aalo [2], are proposed to schedule Coflows of various types, e.g. volume-based Coflows from distributed storage applications vs. streaming-based Coflows from distributed data analysis applications, by adopting a different priority policy for Coflows. These schedulers are designed to coordinate Coflows within *one* network core, and they enforce bandwidth allocation decisions by flow-level transmission rate control at the senders.

Weaver can accommodate a variety of Coflow scheduling policies desired by the system operator, by adopting the corresponding inter-Coflow schedulers, e.g. Varys or Aalo, as the BA in Weaver. Following Weaver's design (Section III), TA decomposes a newly arrived Coflow into K children Coflows, i.e. one child per BA, so that a child Coflow's demand is submitted to and managed by the corresponding BA.

Our objective is to optimize CCTs for *parent* Coflows. Because a parent Coflow is managed by multiple BAs, sharing the status of parent Coflows is necessary for BAs and TA to optimize our objective. Towards this goal, we introduce a Status Server (SS) in Weaver, so each BA periodically reports to SS with the status of children Coflows being managed, revealing status such as residual demand and transmitted bytes. SS aggregates the status of children Coflows to infer the status of their parents.

A. Optimizing BA with SS

The priority ordering of a child Coflow should be determined based on its parent. It is possible for applications to specify the priority of parent Coflows. In the common case of optimizing the average CCT, smaller parent Coflows that are expected to finish sooner should be prioritized in scheduling. However, a smallest-children-Coflows-first policy in each BA does not necessarily lead to our goal of smallest-parent-Coflows-first. For example, consider two Coflows C_1 and C_2 serviced simultaneously by two network cores s_1 and s_2. If BAs on s_1 and s_2 each adopt a different priority ordering for the children of C_1 and C_2, say, BA for s_1 demoting C_1's child while BA for s_2 demoting C_2's child, both Coflows would be delayed due the fact that both Coflows have a low priority child.

To ensure a consistent priority ordering among BAs based on parent Coflows, each BA assigns priority for a child Coflow based on its parent's status queried from SS, and allocates resource for the child based on its actual demand. This approach ensures Coflow children are scheduled towards the goal of optimizing performance for the parents.

B. Optimizing TA with SS

In the presence of multiple Coflows in the system, residual demand from unfinished Coflows should be considered. For example, consider two cores s_1 and s_2 with capacity $r^{(1)}$=1 and $r^{(2)}$=1.5. An active Coflow C_a has residual demand of 3 at $in.1$ of s_2. For a new Coflow C_b with a flow of size 3 from $in.1$, the slower core s_1 turns out to be a more favorable choice for C_b to optimize its CCT when C_a's residual demand is simultaneously considered. In the common case of using smallest-Coflow-first priority to optimize the average CCT, smaller Coflows are prioritized and thus less sensitive to the residual demand. In contrast, large Coflows are more likely to yield to other Coflows, and thus more exposed to the impact from residual demand.

To optimize traffic assignment for Coflows that are most impacted by residual demand, we extend Weaver's TA algorithm as follows. TA classifies a new Coflow to be "large" if the bottleneck of its demand matrix is larger than the bottleneck of the aggregated demand matrix of all unfinished Coflows in the system. To optimize performance for a large Coflow, TA includes the residual demand queried from SS in computing the Coflow's traffic assignment, by initializing a core's assigned demand matrix $\mathbf{D}^{(k)}$ with the core's aggregated residual

demand (line 2). This approach enables TA to adjust its CCT estimation based on the residual demand.

VI. Performance Evaluations

We begin our performance evaluations with testbed experiments. We then extend our evaluations to large-scale simulations to analyze Weaver's performance at scale.

A. Methodology

Workload: We use a realistic workload based on a one-hour MapReduce trace collected from a Facebook production cluster [32]. The trace contains more than 500 Coflows observed in a 150-port fabric with 150 Gbps bisection bandwidth. The Coflows are in various structures (one-to-one, one-to-many, many-to-one and many-to-many).

HPNs configurations: We evaluate a range of configurations of HPNs, as shown in Table IV. The ideal case is a monolithic network ($K = 1$), where all bandwidth is provided by one network core. When $K > 1$, bandwidth is distributed among K cores. For each configuration of HPNs, the bandwidth distribution is displayed in ratios of $\left(\frac{r^{(1)}}{R} \times 100\% : \frac{r^{(2)}}{R} \times 100\% : ...\right)$, where $r^{(k)}$ is the link rate of the k-th core, and $R = \sum_{k=1}^{K} r^{(k)}$ is the sum of link capacity of all K cores. The total ratio of one configuration is 100%. We evaluate configurations at a step of 10% under a specific $K \in \{2, 3, 4\}$. One can map the actual settings of HPNs with our configuration setups by comparing the ratio of bandwidth split. For example, for a 10G/40G HPNs, the results for the 20%:80% split is relevant. Similarly, for a 40G/100G HPNs, 30%:70% split is relevant.

Performance metric: The average CCT is commonly used in prior works [1, 2, 3, 4] to measure Coflow performance when multiple Coflows coexist in the system. As discussed in Section V, scheduling Coflows to minimize the average CCT is an NP-hard problem even in the simplified case of monolithic network (K=1), and heuristic Coflow schedulers are proposed for this simplified case. To quantify the Coflow performance in HPNs ($K \geq 2$), our baseline for comparison is the average CCT in the monolithic network scheduled by Weaver's BA, which is typically a state-of-the-art Coflow scheduler designed to minimize the average CCT in the monolithic network. This baseline assumes an ideal scenario which does *not* involve traffic assignment to different network cores.

Theorem 4.3 considers an initially idle network where there is no waiting time before a Coflow starts. In a real network, CCT is determined by both the waiting time for other Coflows using the network as well as the service time incurred for the Coflow. Thus, the total CCT slowdown of a Coflow tends to be larger than α in Theorem 4.3, which is observed in our experiments, especially for less efficient scheduling schemes.

B. Testbed Evaluations

Implementations: We build a testbed on 30 hosts connected to one Ethernet switch. Each host has 128GB RAM and six 3.5GHz dual hyper-threaded CPU cores. Each physical link is partitioned into K virtual links to emulate K network cores. Traffic congestion is experienced at the ingress and the egress due to the transmission rate control at the senders, and there is no congestion in the Ethernet switch. This behavior is identical to the behavior of HPNs, where congestion is experienced at the ingress/egress but not in the HPNs cores.

TABLE IV: Configuration index of bandwidth distribution under various K. The ideal case is a monolithic network (K=1) providing 100% bandwidth.

Index	K=2	K=3	K=4
1	10%:90%	10%:10%:80%	10%:10%:10%:70%
2	20%:80%	10%:20%:70%	10%:10%:20%:60%
3	30%:70%	10%:30%:60%	10%:10%:30%:50%
4	40%:60%	10%:40%:50%	10%:10%:40%:40%
5	50%:50%	20%:20%:60%	10%:20%:20%:50%
6		20%:30%:50%	10%:20%:30%:40%
7		20%:40%:40%	10%:30%:30%:30%
8		30%:30%:40%	20%:20%:20%:40%
9			20%:20%:30%:30%

We adopt the Coflow scheduling platform in prior work [1] along with its default settings such as coordination intervals. Our baseline is Varys scheduling on the monolithic network. For Weaver that operates on HPNs, we reuse the bandwidth allocation component of Varys as the BAs, and we implement Weaver's TA algorithm.

The original Coflow trace is based on a 150-port fabric with 150 Gbps bisection bandwidth. To match the 30-port with 30 Gbps bisection bandwidth setting in our testbed, we scale down each Coflow by randomly selecting 1/5 of its senders and 1/5 of its receivers and remapping them to our testbed hosts, while preserving the traffic characteristics between the selected senders and receivers. The fraction number of senders (receivers) is rounded to the floor plus 1, so that each Coflow has at least one sender (receiver).

Weaver achieves performance close to the ideal monolithic network for Coflows: Figure 4 highlights that, over a range of HPNs configurations, Weaver achieves consistent performance close to the baseline scheduler that operates in the monolithic network. The normalized average CCT is as low as $1.02\times$ and no larger than $1.1\times$ for the most common case of $K = 2$.

Our testbed results generally resemble those of simulations, which we will show in Section VI-C. Compared to simulations, testbed CCTs are generally longer due to coordination overhead, such as extra delay to distribute control decisions and to adjust flow transmission rate. In addition, such overhead could have a multiplicative effect to prolong CCTs because a delayed Coflow can be further delayed due to bandwidth preemption by newly arrived Coflows.

C. Coflow Scheduling Efficiency

Settings: We implement a flow-level trace-driven discrete-event simulator[2] with various algorithms for traffic assignment and inter-Coflow scheduling. Our simulator performs detailed trace replay based on the realistic Coflow workload.

In the previous Section VI-A, we have introduced one comparison baseline as the Coflow performance under the ideal monolithic network. To understand the Coflow scheduling

[2]https://github.com/sunnyxhuang/weaver

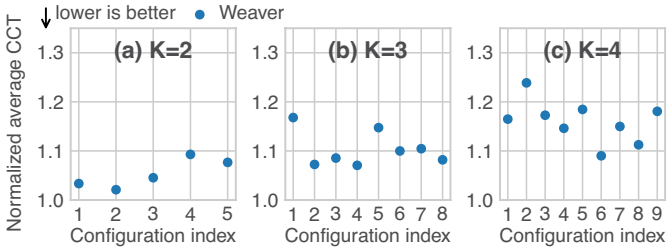

Fig. 4: [Testbed] Average CCT. Normalized over the average CCT in the baseline system that operates on the ideal monolithic network (K=1). Definitions of configuration indexes are in Table IV.

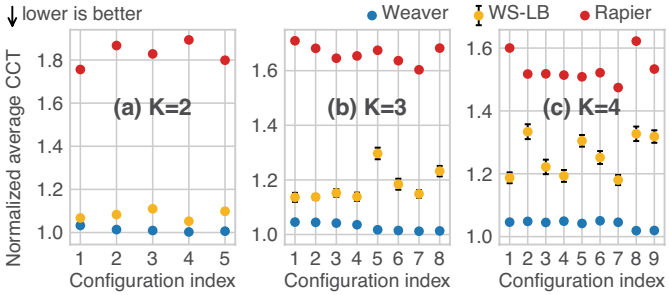

Fig. 5: Average CCT under various K. Normalized over the average CCT in the ideal monolithic network (K=1) under Varys scheduling. Definitions of configuration indexes are in Table IV.

efficiency in HPNs, we adapt two relevant schedulers to be two additional baselines for comparison, as follows.

Additional baseline #1: Weighted Stochastic Load Balance (WS-LB) leverages a weighted random TA algorithm to achieve stochastic load balance among the multiple cores with various capacity in HPNs. Its TA algorithm takes a Coflow demand matrix as the input and assigns a network core to each flow. Each flow is considered independently and the probability that a flow is assigned to a network core is *proportional* to the link capacity of the core. Then the flow is scheduled by the BA of the assigned core. Because of the randomness in its TA algorithm, the WS-LB scheduler is evaluated over 50 runs under each HPNs configuration, and in each run its TA algorithm is initialized with a different random seed.

Additional baseline #2: Rapier [28] is a linear programming (LP) based Coflow scheduler designed to minimize average CCT in a generic topology. To schedule a Coflow, Rapier will (1) determine the flow-level traffic assignment, i.e. the network core assigned for each flow, and then depending on (1) Rapier will (2) calculate the Coflow's bandwidth share, which is further translated to flow-level transmission rate. Both steps rely on an LP solver. These steps are also highly coupled because their combined results further impact the scheduling for other Coflows. In the context of our paper, Rapier controls TA as well as all BAs, and decisions for TA and BAs are mutually dependent.

We have evaluated two types of BAs, i.e. Varys [1] and Aalo [2], which are two state-of-the-art Coflow schedulers both designed to minimize the average CCT. Varys is designed to optimize performance for volume-based Coflows, while Aalo is designed to service streaming-based Coflows. The default type of BA is Varys for Weaver and WS-LB. Rapier has highly coupled its traffic assignment and scheduling policy, so Rapier is evaluated as a whole.

Weaver achieves Coflow performance comparable to the ideal monolithic network. We start by comparing all schedulers under the default settings. Figure 5 highlights that Weaver's average CCT is comparable to that of the ideal monolithic network, across a range of HPNs configurations. We observe Weaver is at most $1.03\times$ of the ideal for K=2 and $1.05\times$ for K=3 or K=4. In contrast, WS-LB performs worse due to its inefficiency in traffic assignment, as we will discuss in Section VI-D.

Compare with the competitive schemes, Rapier is significantly worse due to its Coflow scheduling policy. To optimize the average CCT, both Rapier and Weaver prioritize the smaller Coflows that are expected to finish sooner. Weaver tends to be more strict in observing the priority, while Rapier may not fully observe the priority order. In the second step of Rapier to decide the bandwidth share for a Coflow by solving a LP problem, if a Coflow contends with higher priority Coflows, the Coflow is forced to pause because the LP solution indicates the Coflow will never finish. As a result, residual bandwidth goes to lower priority Coflows, rather than the paused Coflows with higher priority, which prolongs the CCTs of paused Coflows. On the other hand, lower priority Coflows can hardly benefit from the bandwidth leftover, because they must yield to the previously-paused higher-priority Coflows when the contention is resolved. Both effects combined, Rapier's scheduling policy becomes significantly inefficient.

D. Optimality of Traffic Assignment

As we have shown in Section IV-A, the traffic assignment problem (Table III) is NP-hard. Theorem 4.3 proves that, in HPNs where all parallel networks are not identical, the optimum approximation ratio of Weaver's TA algorithm is always better than that of Rapier or WS-LB. In the common case of $K = 2$, Table V compares the approximation ratio of various TA algorithms.

We demonstrate how Weaver achieves better performance guarantee with an example in Figure 6, where the traffic demand of an incast Coflow (Figure 6a) is to be assigned to two network cores with link capacity $r^{(1)} = 1$ and $r^{(2)} = 4$ respectively. The CCT of the incast Coflow is determined primarily by the most congested port $out.4$.

Weaver benefits from the critical flow classification (Algorithm 1 line 4), so that critical flows that are more likely to prolong CCT are assigned to the less congested network core to speed up the Coflow. While Weaver seeks utilizing bandwidth from multiple network cores to speed up a Coflow, the algorithm guarantees to be no worse than assigning all traffic to the fastest core. In this example, all flows from the incast Coflow are critical because they impact CCT on $out.4$. By measuring the children CCTs (line 5), Weaver decides to assign all flows to the faster core s_2, as shown in Figure 6b.

On the other hand, WS-LB could significantly delay a Coflow by assigning all traffic to the slowest core due to

TABLE V: Approximation ratio of various traffic assignment algorithms under K=2. Lower values indicate better performance guarantee with closer approximation to the optimum.

	Weaver	WS-LB	Rapier
10%:90%	1.11×	10.0×	2×
20%:80%	1.25×	5.00×	2×
30%:70%	1.43×	3.33×	2×
40%:60%	1.67×	2.50×	2×

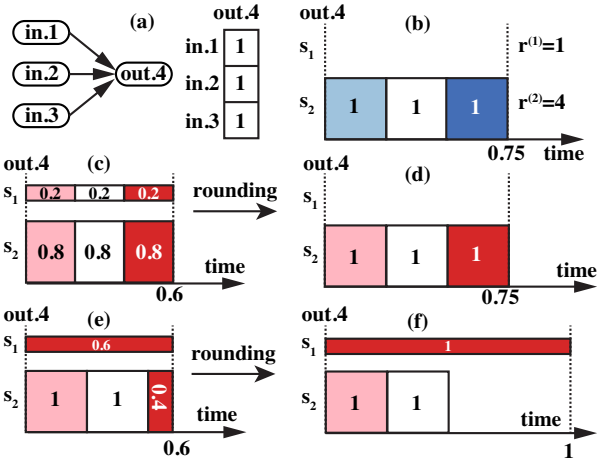

Fig. 6: Traffic assignment of one Coflow to two network cores, s_1 and s_2, with link capacity $r^{(1)} = 1$ and $r^{(2)} = 4$. (a) Traffic demand for an incast Coflow, whose CCT is determined by $out.4$. (b-d) show the timeline and traffic load on out.4 for s_1 and s_2. (b) Weaver's TA. (c, e) Solution of the relaxed problem allowing flow splitting. A flow may split among two cores. (d, f) Solution after rounding to avoid flow splitting. The split flow is assigned to the core that takes up more portion of the flow in the solution of relaxed LP.

randomness. We omit this case in Figure 6 to simplify representation. For the LP-based Rapier algorithm, its inefficiency stems from the *alternate optima under problem relaxation*. As discussed in Section IV-A, our problem without flow splitting (Table III) can not be efficiently solved by LP techniques. Hence, an LP-based algorithm must start from a relaxed problem and round the solution to satisfy the broken constraints. Rapier relaxes the problem by allowing flow spitting. The relaxed problem is likely to have multiple optimal solutions, as shown in Figure 6c and Figure 6e. However, the optimal solutions of the relaxed problem are *not* equally efficient after rounding. For example, while one solution (Figure 6c) produces an efficient traffic assignment as Weaver's TA algorithm, the other solution (Figure 6e) mistakenly assigns one flow to the slower core and delays the Coflow. An optimal solution of the relaxed problem can not guarantee the efficiency of its rounded result, and thus Rapier fails to achieve the performance guarantee as good as Weaver's TA algorithm.

E. Sensitivity Analysis

We also evaluate Weaver's robustness over various factors. **Sensitivity to assignment ordering.** To evaluate how the ordering of traffic assignment affects Coflow performance, we conduct a sensitivity test. In this test, we change the

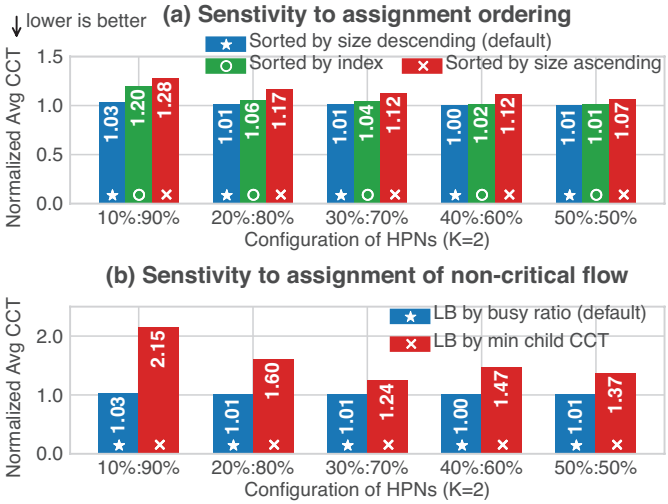

Fig. 7: Sensitivity to (a) assignment ordering and (b) assignment of non-critical flows. The presented average CCT is normalized over the average CCT in the ideal monolithic network (K=1) under Varys scheduling.

assignment ordering by sorting flows in the following ways (line 3 in Algorithm 1): (1) descending sort by flow sizes, which is used as the default, (2) ascending sort by tuple of input and output port index, and (3) ascending sort by flow sizes. As shown in Figure 7a, we observe prioritizing larger flows in assignment is the most efficient. In contrast, the reversed ordering that considers smaller flows first is the least efficient. This confirms our previous observation that larger flows are more likely to determine CCT and thus they should receive priority to take up faster network cores. As expected, when the capacity of network cores becomes uniform, the performance gap resulted from various ordering becomes smaller, because the penalty of mismatching flow sizes and core capacity becomes smaller.

Sensitivity to assignment of non-critical flows. We conduct another sensitivity test to evaluate how the load balancing (LB) technique for non-critical flows impact on Coflow performance. In this test, we change the assignment of non-critical flows (line 7) by assigning the flow to the network core (1) which has the minimum busy ratio (called *LB by busy ratio*, used as the default), and (2) which has the minimum CCT for the flows assigned on the core from the Coflow in consideration (called *LB by min child CCT*). Note that (2) is equivalent to reusing the assignment strategy of *critical* flows (line 5) for *non-critical* flows. We found *LB by busy ratio* is the most efficient in various cases. *LB by min child CCT* is the least efficient because the faster cores tend to be overloaded due to bias of the bottleneck, as discussed in Section IV-C.

Weaver remains robust under different scheduling policies. To understand how Weaver performs with another Coflow scheduling policy, we change all BAs from Varys to Aalo. Under Weaver with Aalo (or Varys) scheduling, for HPNs configurations of 10%:90%, 20%:80%, 30%:70%, 40%:60%, and 50%:50% respectively, CCT is on average 1.08× (or 1.07×), 1.15× (or 1.13×), 1.20× (or 1.18×), 1.23× (or

1.21×), and 1.29× (or 1.28×) of the CCT in the ideal monolithic network with Aalo (or Varys) scheduling. The CCT slowdown with Aalo scheduling is comparable to that with Varys when moved from the ideal monolithic network to HPNs in a range of settings. In summary, Weaver remains robust from Varys to Aalo scheduling.

VII. Related Work

Coflow scheduling in data center networks: A range of recent works [1, 2, 3, 4] have demonstrated the benefits of leveraging application-level traffic requirements, expressed in Coflows, to improve application-level communication performance. These inter-Coflow schedulers [1, 2, 3, 4] are limited by their over-simplified assumption that abstracts the whole network fabric as *one* network core. Our work is complementary to these existing works in Coflow scheduling, so that Coflow traffic is assigned to network cores in a way that allows efficient inter-Coflow scheduling in each individual core. Rapier [28] is proposed to schedule Coflows in a generic topology, but Rapier is not efficient in HPNs due to a range of factors as discussed in Section VI.

Heterogeneous parallel networks (HPNs): We have introduced HPNs in data centers in Section I and Section II. In the context of wireless networks, HPNs is also relevant because multiple wireless networks usually overlap in space for users to access simultaneously, such as Wifi and cellular networks. Recent works have studied how to exploit such wireless networks in parallel to support web content delivery [33] and video streaming [34]. Exploiting heterogeneous parallel wireless networks is fundamentally different from our problem settings. For example, traffic in wireless networks tends to be simplex point-to-point flows to service web and video applications, while traffic in data centers often comes in structured flows from the distributed data-parallel applications. As for design goals, wireless networks often stress on latency and energy efficiency to service mobile users in high-interference environments, while we aim at application-level performance and incremental network evolutions to achieve higher performance at lower cost for data center networks.

Flow-level load balancing schemes: ECMP [35] and MPTCP [36] are commonly known techniques for transmission over multiple alternative paths. They aim at load balancing among alternative paths to maximize bandwidth utilization. These schemes assume bandwidth fair sharing among contending flows. However, they are incapable of optimizing Coflow level scheduling objectives such as reducing the average CCT, and flow level fair sharing is known to result in poor Coflow performance [1]. In contrast, our work considers the assignment and scheduling of Coflows to improve application performance in HPNs.

VIII. Conclusions

We present Weaver, the first scheduler to service Coflows in HPNs with high application level communication efficiency. Weaver leverages an efficient traffic assignment algorithm which is proven to be within a constant factor of the optimal. Weaver also serves as a framework to accommodate a variety of traffic scheduling policies to improve performance at the application level. The Weaver-orchestrated HPNs achieve Coflow performance comparable to the ideal monolithic network. As incremental upgrades of infrastructure becomes the trend, our work demonstrates how an evolving data center can make the most out of its multiple generations of network fabrics.

Appendix

Proof of Theorem 4.1 We prove by contradiction. Assume $\forall k : \frac{L(\mathbf{D}^{(k)})}{r^{(k)}} < \frac{L(\mathbf{D})}{R}$. Then we have $\forall k : L(\mathbf{D}^{(k)}) < \frac{r^{(k)}}{R} L(\mathbf{D})$. Without loss of generality, assume the m-th row in $\mathbf{D}$ yields $L(\mathbf{D})$, i.e. $\sum_{j=1}^{N} d_{m,j} = L(\mathbf{D})$. Therefore, we have

$$\forall k : \sum_{j=1}^{N} d_{m,j}^{(k)} \leq L(\mathbf{D}^{(k)}) < \frac{r^{(k)}}{R} \sum_{j=1}^{N} d_{m,j}. \tag{1}$$

Adding up all K equations for $k \in \{1, .., K\}$ in Equation (1),

$$L(\mathbf{D}) = \sum_{j=1}^{N} d_{m,j} = \sum_{k=1}^{K} \sum_{j=1}^{N} d_{m,j}^{(k)}$$
$$< (\frac{r^{(1)}}{R} + ... + \frac{r^{(K)}}{R}) \sum_{j=1}^{N} d_{m,j} = \frac{\sum_{k=1}^{K} r^{(k)}}{R} L(\mathbf{D}) = L(\mathbf{D})$$

results in a contradiction of $L(\mathbf{D}) < L(\mathbf{D})$. Similarly, when $L(\mathbf{D})$ is given by a column of D, a contradiction is derived by replacing the row sum with column sum in Equation (1). These complete the proof by contradiction. ■

Proof of Theorem 4.2 We begin by proving a proposition

$$\forall k_1, k_2 : \frac{L\left(\mathbf{D}^{(k_1)}\right)}{r^{(k_1)}} \leq \frac{L(\mathbf{D})}{r^{(k_2)}}.$$

When $k_1 = k_2$, our proposition is true because $L(\mathbf{D}^{(k_1)}) \leq L(\mathbf{D})$. When $k_1 \neq k_2$, we prove our proposition by contradiction. Assume $\frac{L\left(\mathbf{D}^{(k_1)}\right)}{r^{(k_1)}} > \frac{L(\mathbf{D})}{r^{(k_2)}}$, then there must be an iteration in Algorithm 1 line 3 where the traffic demand of $d_{i,j}$ from flow $f_{i,j}$ is assigned to the k_1-th network core to make up $\mathbf{D}^{(k_1)}$. Denote $\mathbf{D}'^{(k)}$ as the traffic demand allocated to the k-th core before adding $f_{i,j}$, and $\mathbf{D}^{(k)} = \mathbf{D}'^{(k)} \cup d_{i,j}$. Because $d_{i,j}$ is added to the k_1-th core, we should have $\forall k_2 \neq k_1 : \frac{L\left(\mathbf{D}^{(k_1)}\right)}{r^{(k_1)}} = \frac{L\left(\mathbf{D}'^{(k_1)} \cup d_{i,j}\right)}{r^{(k_1)}} \leq \frac{L\left(\mathbf{D}'^{(k_2)} \cup d_{i,j}\right)}{r^{(k_2)}} \leq \frac{L(\mathbf{D})}{r^{(k_2)}}$, which results in a contradiction with our assumption.

Thus, our proposition is proved and Theorem 4.2 is an immediate result of our proposition. ■

Proof of Theorem 4.3 We know $T_o^s \leq T_o^f$. Theorem 4.1 shows $T_o^s = \frac{L(\mathbf{D})}{\sum_{k=1}^{K} r^{(k)}}$ and Theorem 4.2 shows $T \leq \min_{\forall k}(\frac{L(D)}{r^{(k)}}) = L(\mathbf{D}) / \max_{\forall k} r^{(k)}$. Therefore, Theorem 4.3 is proved because

$$T/T_o^s \leq \frac{\sum_{k=1}^{K} r^{(k)}}{\max_{\forall k} r^{(k)}} = \frac{r^{(1)}}{\max_{\forall k} r^{(k)}} + ... + \frac{r^{(K)}}{\max_{\forall k} r^{(k)}} \leq K. \blacksquare$$

ACKNOWLEDGMENT

We thank the BOLD Lab memrbers and the anonymous reviewers for useful feedback. This research is sponsored by the NSF under CNS-1422925, CNS-1718980, CNS-1801884, and CNS-1815525.

REFERENCES

[1] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with Varys," in *ACM SIGCOMM*, 2014.
[2] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *ACM SIGCOMM*, 2015.
[3] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward automatically identifying and scheduling coflows in the dark," in *ACM SIGCOMM*, 2016.
[4] X. S. Huang, X. S. Sun, and T. S. E. Ng, "Sunflow: Efficient optical circuit scheduling for coflows," in *ACM CoNEXT*, 2016.
[5] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *ACM SIGCOMM*, 2018.
[6] Apache Hive. http://hive.apache.org.
[7] Apache HDFS. https://hortonworks.com/apache/hdfs/.
[8] Apache Hadoop. http://hadoop.apache.org/.
[9] Apache Spark. https://spark.apache.org/.
[10] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, "Altruistic scheduling in multi-resource clusters," in *USENIX OSDI*, 2016.
[11] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *ACM SIGCOMM*, 2013.
[12] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2008.
[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*, 2009.
[14] Cisco. (2016) Cisco global cloud index: Forecast and methodology, 2015–2020. https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf.
[15] The Register. (2012) Network boffins say terabit Ethernet is too fast. https://www.theregister.co.uk/2012/09/28/terabit_ethernet_too_fast/.
[16] The Register. (2013) Forget terabit Ethernet, the next step is 400 gig. https://www.theregister.co.uk/2013/11/21/forget_terabit_ethernet_the_next_step_is_400_gig_if_we_can_afford_it/.
[17] Wikipedia. (2017) Terabit Ethernet. https://en.wikipedia.org/wiki/Terabit_Ethernet.
[18] Cisco White Paper. (2016) The future is 40 Gigabit Ethernet. https://www.cisco.com/c/dam/en/us/products/collateral/switches/catalyst-6500-series-switches/white-paper-c11-737238.pdf.
[19] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *ACM SIGCOMM*, 2015.
[20] V. Liu, D. Zhuo, S. Peter, A. Krishnamurthy, and T. Anderson, "Subways: A case for redundant, inexpensive data center edge links," in *ACM CoNEXT*, 2015.
[21] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *USENIX NSDI*, 2010.
[22] I. W. Group. (2012) IEEE 802.3 industry connections bandwidth assessment. http://www.ieee802.org/3/ad_hoc/bwa/BWA_Report.pdf.
[23] A. F. Bach. (2011) Bandwidth demand in the financial bandwidth demand in the financial industry: The growth continues. http://www.ieee802.org/3/ad_hoc/bwa/public/jun11/bach_01a_0611.pdf.
[24] (2018) Price for off-the-shelf transceivers. http://www.fs.com.
[25] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.* (2014) Introducing data center fabric, the next-generation facebook data center network. https://code.facebook.com/posts/360346274145943.
[26] The CTO Advisor. (2016) 25Gbps vs. 40Gbps Ethernet. https://www.thectoadvisor.com/blog/2016/12/14/25gbps-vs-40gbps-ethernet.
[27] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM CCR*, 2008.
[28] 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 *IEEE INFOCOM*, 2015.
[29] J. Bruno, E. G. Coffman Jr, and R. Sethi, "Scheduling independent tasks to reduce mean finishing time," *Communications of the ACM*, 1974.
[30] E. Horowitz and S. Sahni, "Exact and approximate algorithms for scheduling nonidentical processors," *Journal of the ACM (JACM)*, 1976.
[31] T. Gonzalez, O. H. Ibarra, and S. Sahni, "Bounds for LPT schedules on uniform processors," *SIAM Journal on Computing*, 1977.
[32] Coflow benchmark based on Facebook traces. https://github.com/coflow/coflow-benchmark.
[33] B. Han, F. Qian, S. Hao, and L. Ji, "An anatomy of mobile web performance over multipath tcp," in *ACM CoNEXT*, 2015.
[34] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan, "MP-DASH: Adaptive video streaming over preference-aware multipath," in *ACM CoNEXT*. ACM, 2016.
[35] C. E. Hopps and D. Thaler, "Multipath issues in unicast and multicast next-hop selection," in *RFC 2991*, 2000.
[36] M. Handley, O. Bonaventure, C. Raiciu, and A. Ford, "Tcp extensions for multipath operation with multiple addresses," in *RFC 6824*, 2013.