Adia: Achieving High Link Utilization with Coflow-Aware Scheduling in Data Center Networks

Jingjie Jiang, Shiyao Ma, Bo Li, *Fellow, IEEE,* Baochun Li, *Fellow, IEEE*

Abstract—Link utilization has received extensive attention since data centers become the most pervasive platform for data-parallel applications. A specific job of such applications involves communication among multiple machines. The recently proposed *coflow* abstraction depicts such communication through a group of parallel flows, and captures application performance through corresponding communication requirements. Existing techniques to improve link utilization, however, either restrict themselves to achieving work conservation, or merely focus on flow-level metrics and ignore coflow-level performance. In this paper, we address the coflow-aware scheduling problem with the objective of maximizing link utilization. Through theoretic analyses, we formulate the coflow-aware scheduling problem as a NP-hard *open shop scheduling problem with heterogeneous concurrency*. We design *Adia*, a hierarchical scheduling framework to conduct both inter- and intra- link scheduling. The design of *Adia* leverages priority-based scheduling while guarantees work-conserving and starvation-free bandwidth allocation at the same time. We also prove *Adia*'s algorithm is 2-approximate in terms of link utilization. Extensive simulation results on ns3 further show that *Adia* outperforms both per-flow mechanisms coflow schemes in terms of link utilization, and achieves similar coflow performance in comparison with the state-of-art coflow scheduling schemes.

Index Terms—Datacenter Networks, Parallel Computing, Big Data Processing, Resource Utilization.

1 Introduction

Achieving high link utilization has come into the academic spotlight since data centers become the *de facto* computing platform for data-parallel applications [1]. As reported in [2], network devices in data centers, such as links and switches, contribute to about 15% of the overall cost. With the expensive devices already deployed, the *average* utilization of network links is unfortunately very low in most data centers. Less than 40% of link bandwidth at the edge layer is utilized on average [1] [3]. However, edge links still suffer from severe congestions during peak hours. The high peak-to-average ratio forces the overs network bandwidth to guarantee the acceptable end-to-end delay even during the peak hours.

To improve the average link utilization, existing flow scheduling mechanisms (*e.g.*, D3 [4], PDQ [5] and pFabric [6]) strive to be work-conserving. Work conservation, however, only tries to improve the utilization of each single link without considering the global situation. Without proper coordination, a flow might unnecessarily occupy the bandwidth on a receiver's downlink, hindering other flows on the same link. Consequently, the uplinks of these affected flows are at risks of underutilization. In this case, the work conservation is not violated, but the link utilization is suboptimal. Scheduling mechanisms focusing on work conservation are thus insufficient to achieve optimal link utilization. To make things worse, tenants of a data center may hide their true traffic demands and throttle some of their data transfers on purpose to acquire more bandwidth on a congested link [7]. The data center operator would then deem that the network is work conserving since the tenants have no more data to send, while the actual link utilization is very low.

Since data-parallel jobs (*e.g.*, [8], [9], [10]) are usually network-bounded [11], the response times depend on network transfers within each job. Such job-specific communication usually involves multiple parallel flows to transmit data among groups of machines in successive computation stages [11]. The recently proposed *coflow* abstraction [12] [13] formally defines a group of concurrent flows that transfer intermediate results among multiple machines as a coflow. A coflow is not considered completed until the completion of *all* its constituent flows. Since the flows in a coflow may have varying sizes and transmission rates, their completion times may fluctuate severely. Given the correlation of flows in a coflow, even if flow-level scheduling schemes can effectively maximize link utilization, their ignorance to coflow-level network requirements would hurt coflow performance.

In this paper, we try to maximize the utilization of access links without sacrificing coflow performance. Specifically, there are three major challenges to this problem. *Firstly*, the utilizations of access links are correlated with each other. The utilization of the uplink of a flow's source depends on the state of the downlink of the same flow's destination. We need to properly coordinate the coupled bandwidth among all uplinks and downlinks to achieve high link utilization. *Secondly*, high link utilization must be achieved without sacrificing coflow performance. As demonstrated in previous coflow scheduling schemes [12] [13], coflow-level priority scheduling effectively reduced the average coflow completion time. Nevertheless, their inter- and intra-coflow scheduling hierarchy may conflict with our link-oriented scheduling objective. A new scheduling framework is needed to consider both link and coflow efficiency. *Thirdly*, coflow-aware scheduling must work in an online fashion. This excludes traditional scheduling wisdom in operation research and combinatorics [14], which need complete information of all the coflows in a network.

- *Jingjie Jiang, Shiyao Ma and Bo Li are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. Their email addresses are* {jjiangaf, smaad, bli}@cse.ust.hk.
- *Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto. His email address is* bli@ece.toronto.edu.

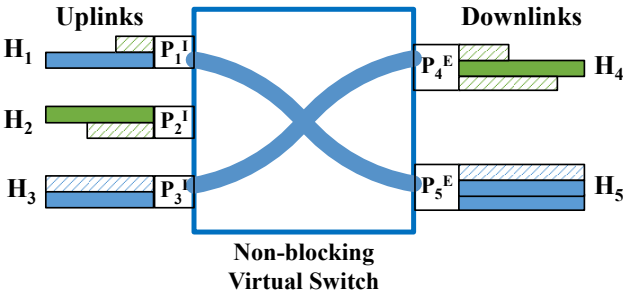


Fig. 1. A logic view of a data center network: machines are inter-connected through a non-blocking switch; congestions only happen at access links.

We propose to address these three challenges by designing *Adia*, a two-level online scheduling framework. The upshot of this paper revolves around the focus on link utilization with coflow awareness. Based on an in-depth analysis of this problem, we prove it is NP-hard by reducing the *open shop scheduling* problem to it. Despite the hardness of this problem, we design a *hierarchical* scheduling mechanism to conduct both inter- and intra- link scheduling. At the first level, *Adia* treats one uplink as an entity, and conducts priority-based inter-link scheduling to reduce the makespan of the schedule. At the second level, *Adia* zooms in to flows belonging to the same uplink, and performs intra-link scheduling. To minimize the average coflow completion time, we prioritize the flows belonging to a faster coflow on an uplink. As reducing coflow completion times essentially enables more concurrent coflows in the network, intra-link scheduling essentially improves network throughput, and henceforth reinforces the performance of inter-link scheduling. As a result, both the average and tail completion time of all coflows effectively decrease. We further prove our algorithm is 2-approximate when all access links are identical.

In production data centers, however, link capacities are non-uniform and even dynamic in case of link failures. We turn to evaluate the performance of *Adia* using *ns3* [15]. The extensive simulation results verify that *Adia* also achieves high link utilization in online scenarios. Meanwhile, the coflow completion time decreases significantly compared to flow-level scheduling schemes, and is comparable to Varys [12], the state-of-art coflow scheduling scheme.

The roadmap of this paper is as follows: we analyze the coherence and conflict between link utilization and coflow performance in Sec. 2. We proceed to theoretically analyze and formulate our problem in Sec. 3 present the design of *Adia* in Sec. 4. We evaluate *Adia*'s performance in Sec. 5. We further discuss our contribution in the context of related work in Sec. 6 before we conclude this paper in Sec. 7.

2 Motivation and Background

2.1 System Model

According to the statistics collected in production data centers [16], core networks seldom experience severe and persistent congestion, while network edges are often congested. Given such observations, we suppose congestions only occur at network edges for simplicity. In other words, the possible congestion locations are ingress queues at a sender's NIC (network interface card) and the egress queues at a receiver's ToR (top-of-rack) switch. The whole data center fabric can then be viewed as a non-blocking switch as shown in Fig. 1. Under such a network model, the only scarce network resource is the bandwidth of access links (*i.e.*, uplinks and downlinks). Therefore, routing strategies that aim at load balancing have no benefits since the access links cannot be bypassed. Furthermore, the specific network topologies (*e.g.*, Fat-tree [17], Bcube [18], Dcell [19] and JellyFish [20]) have no influence on scheduling strategies.

TABLE 1
Notations and Definitions

Notation	Definition
M	the number of physical machines
P_i^I, P_i^E	the ingress and egress ports of P_i
B_i	the total bandwidth of port P_i
B_i^I, B_i^E	the ingress and egress bandwidth of P_i
$C_k = \{f_{ij}^k\}$	a coflow consisting of many concurrent flows
f_{ij}^k	a flow from P_i to P_j belonging to C_k
T_k	the completion time of coflow C_k
$M_k = [s_{ij}^k]$	the traffic matrix of coflow C_k
$W_k = [w_{ij}^k]$	the amount of transmitted data of flows in C_k
$L = \max T_k$	the makespan of a feasible schedule
s_{ij}	the amount of traffic from P_i to P_j
s_{ij}^k	the amount of traffic from P_i to P_j belonging to C_k
r_{ij}	the aggregate rates from P_i to P_j
r_{ij}^k	the rate of flow f_{ij}^k
t_{ij}^k	the completion time of flow f_{ij}^k

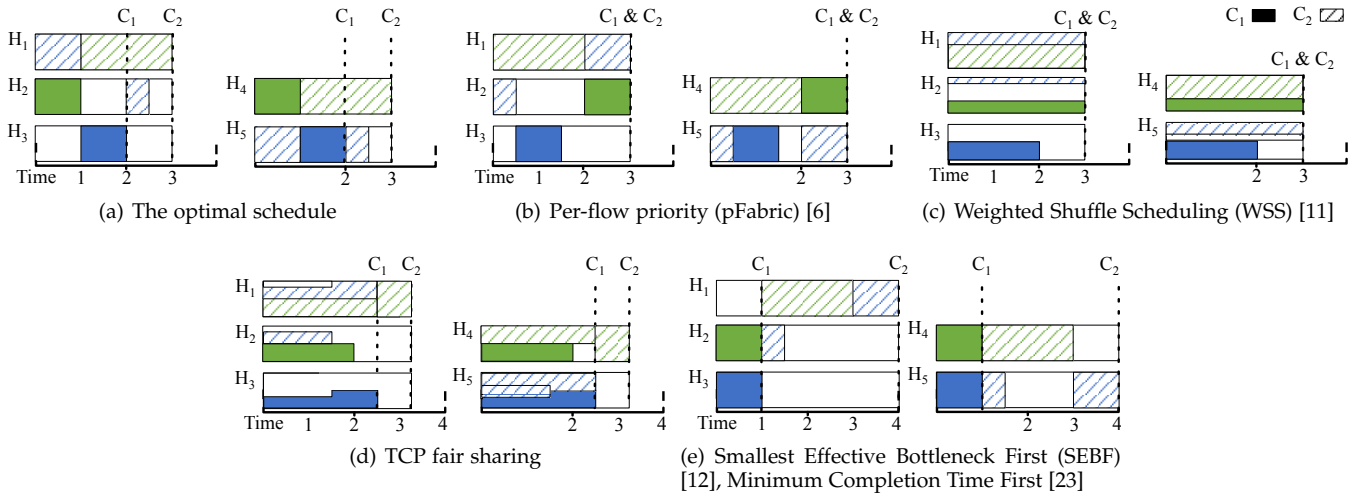
Existing coflow-aware scheduling schemes simply aim to reduce the average coflow completion time. Intuitively, reducing coflow completion times brings about higher link utilization. We contend that rather than the average coflow completion time, the key to maximizing link utilization is when the last coflow finishes. In other words, the makespan of a schedule is critical for improving network efficiency.

One may argue that a link can be disabled to save power when it is idle and thus the idle time should not be considered when optimizing link utilization. However, a temporary idle link does not indicate that there is no pending traffic on that link which needs to be transferred very soon. For instance, in Fig. 2(a), H_2 is idle at time 1 but it needs to transfer data at time 2. Turning down such links will either at the risk of harming network availability or incur too frequent on-offs, which consume even more power. Furthermore, even if an idle link currently has no pending traffic demand, we do not know if there will be more coflows to arrive in online systems since the execution time of a computation job is often non-deterministic [21]. Without priori information about when coflows arrive, turning down idle links will harm network connectivity. Therefore, we contend that idle links have to remain online as in [1] [22]. Possible energy saving schemes are beyond the scope of this paper.

2.2 Motivating Example

Reducing average coflow completion times as in existing coflow-aware scheduling schemes [12] [13] does not necessarily indicate higher link utilization. We analyze the relationships among work conservation, link utilization and coflow completion times through an example shown in Fig. 2.

A coflow C_k is depicted through a traffic matrix $M_k = [s_{ij}^k]$, where s_{ij}^k equals to the amount of traffic a flow in C_k sent from



(a) The optimal schedule (b) Per-flow priority (pFabric) [6] (c) Weighted Shuffle Scheduling (WSS) [11]

(d) TCP fair sharing (e) Smallest Effective Bottleneck First (SEBF) [12], Minimum Completion Time First [23]

Fig. 2. A motivating example to discuss the relation among work conservation, link utilization and coflow completion times: the flows of both coflows arrive at time 0. All the schedules are work-conserving. The per-flow fair sharing and SEBF are suboptimal in terms of link utilization; per-flow sharing, per-flow priority and WSS are suboptimal in terms of the average coflow completion time. Flows sent to H_4 are green and sent to H_5 are blue.

the ingress port P_i^I to the egress port P_j^E. The traffic matrices of the two coflows used in the example are shown below:

$$M_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \; M_2 = \begin{pmatrix} 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Three hosts, H_1, H_2 and H_3, initiate five flows to the two receivers, H_4 and H_5, in the network. Suppose each link at most transmits one unit of data in one time unit. We show different scheduling outputs under various strategies in Fig. 2.

It is easy to verify that all the five schedules are work-conserving, but the link utilization and coflow completion times vary significantly. The per-flow fair sharing strategy equally shares the bandwidth among all the flows, and achieves the results in Fig. 2(d): the two coflows finish within 2.5 and 3.25 time units, and the average link utilization is 67.7%. pFabric [6] prioritizes flows with smaller sizes. The best performance achieved is shown in Fig. 2(b) with completion times of both coflows equal to 3 time units and the average link utilization equal to 73.3%. A weighted sharing strategy proposed in Orchestra [11] aims to reduce coflow completion times but only achieves the same performance as pFabric. In contrast, the smallest effective bottleneck first algorithm in Varys [12] and the minimum completion time first strategy in Rapier [23] trims the average coflow completion time to 2.5. Essentially, since no alternate path can bypass the access links, Rapier is equivalent to Varys under such circumstances. Nevertheless, such a strategy prolongs C_2, and thus brings downs the average link utilization to 55%. The optimal schedule in Fig. 2(a) achieves the same average coflow completion time but raises the average link utilization to 73.3%.

We obtain three key observations through this example. First, the makespan of all coflows, rather than their average completion time, determines link utilization. Second, achieving work conservation is *necessary yet insufficient* to maximize link utilization. Finally, among all the schedules that can maximize the link utilization, a schedule that considers coflow semantics can further improve average coflow performance. Therefore, we need to consider link utilization and coflow completion time at the same time to optimize these two important performance metrics for data-parallel jobs.

3 Problem Formulation

Given a set of coflows running in data centers, we try to schedule flows belonging to different coflows with the objective of maximizing link utilization. Suppose there are N coflows transferring data among M physical servers in a data center network. We suppose all coflows arrive at the same time for the ease of analysis, but our conclusions can be easily applied to coflows with different arrival times. Define a coflow as $C_k = \{f_{ij}^k\}$, where the flow f_{ij}^k transfers s_{ij}^k amount of data from an ingress port P_i^I to an egress port P_j^E. We only consider single-wave coflows like in existing coflow scheduling schemes [12], [24], [25], namely, all the flows belonging to the coflows arrive at the same time. The structure of a coflow C_k can be depicted through a traffic matrix $M_k = [s_{ij}^k]_{P \times P}$. Since a coflow is considered completed only when all its constituent flows finish, the coflow completion time can be represented as

$$T_k = \max_{f_{ij}^k \in C_k} \frac{s_{ij}^k}{r_{ij}^k}, \; \forall k \tag{1}$$

$$r_{ij}^k = \frac{1}{t_{ij}^k} \int_0^{t_{ij}^k} r_{ij}^k(t)\, dt \tag{2}$$

r_{ij}^k is the average rate of a flow through its lifetime and t_{ij}^k is the flow duration time. Replacing the variables of r_{ij}^k with the expression (2) yields

$$\int_0^{T_k} r_{ij}^k(t)\, dt = s_{ij}^k, \; \forall i, \forall j \tag{3}$$

Eq. (3) guarantees that any flow belonging to C_k is able to transmit all its data within T_k time units. Let L denote the makespan (or the length) of a schedule. Essentially, L equals to the maximum completion times of all the coflows:

$$L = \max_k T_k \tag{4}$$

Eq. (4) indicates that all the coflows have completed transmission after L time units. Therefore, we can derive that

$$s_{ij} = \sum_k s_{ij}^k = \int_0^L r_{ij}(t)\,dt,\ \forall i,\ \forall j \tag{5}$$

$$\text{where}\ \ r_{ij}(t) = \sum_k r_{ij}^k(t) \tag{6}$$

To find a feasible schedule, any access link should not be overloaded. In other words, the sum of flow rates on a given link cannot exceed its capacity. The bandwidth of a port P_i, denoted as B_i is divided into the ingress bandwidth B_i^I and egress bandwidth B_i^E. The capacity constraints of uplinks and downlinks are shown in Eq. (8) and (9). Together with the constraint (3), the problem of maximizing the aggregated utilization of all access links can be formulated as

$$\max \quad \frac{1}{L}\int_0^L \frac{\sum_i\sum_j (r_{ij}(t)+r_{ji}(t))}{\sum_i (B_i^I + B_i^E)}\,dt \tag{7}$$

$$\text{s.t.} \quad \sum\nolimits_j r_{ij}(t) \le B_i^I, \qquad \forall t,\ \forall i \tag{8}$$

$$\sum\nolimits_j r_{ji}(t) \le B_i^E, \qquad \forall t,\ \forall i \tag{9}$$

Since the egress and ingress bandwidths add up to a port's total bandwidth, Eq. (7) can be transformed to

$$\frac{1}{L}\frac{\sum_i\sum_j \int_0^L (r_{ji}(t)+r_{ij}(t))\,dt}{\sum_i B_i} \tag{10}$$

Replacing the integral term with s_{ij} as in Eq. (5) yields

$$\frac{1}{L}\frac{\sum_i\sum_j (s_{ji}+s_{ij})}{\sum_i B_i} \tag{11}$$

Since B_i, s_{ij} and s_{ji} are fixed, minimizing the makespan of all coflows can maximize the average link utilization. This coincides with our previous observation. When flows are bottlenecked at the receiver side, some bandwidth of a sender's uplink would be left unusable. The effective capacity of an uplink is thus restricted by the load of the corresponding downlinks. In other words, the capacities of uplinks and downlinks are *interdependent*. By regarding uplinks as jobs to be scheduled and downlinks as machines with nonuniform capabilities, we formulate the scheduling problem with the objective (11) as a variant of *open shop scheduling problem* [26]. The dependencies among access links can then be transformed to *heterogeneous concurrency* constraints (see Appendix A). It is worth noticing that the uplinks and downlinks are *interchangeable* in our problem. Whatever has been proved for uplinks as jobs and downlink as machines can also be proven for downlink as machines and uplink as jobs.

Furthermore, the coflow performance is measured through the average coflow completion time (CCT):

$$\min \quad CCT = \frac{1}{n}\sum_{k=1}^{n} T_k \tag{12}$$

$$\text{s.t.} \quad (3)\ (8)\ (9)\ \text{hold}$$

We demonstrate that minimizing the makespan or average coflow completion time with coupled link resources is NP-hard even when: 1) all coflows start at the same time with full knowledge of their constituent flows (*i.e.*, offline cases); and 2) ingress and egress ports have the same capacity. We prove the NP-hardness of our scheduling problem in Appendix A. As a sketch of proof, the NP-complete *open shop scheduling problem* [26] can be reduced to our problem with the objective of minimizing the makespan; the NP-hard concurrent open shop scheduling problem [27] can be reduced to our problem with the objective of minimizing average coflow completion time. Since the correlation between the average and tail completion time is unnecessarily positive, it is not always possible to find an optimal schedule that both minimizes the average completion time and the makespan. A trade-off between the coflow performance and link utilization needs to be made when scheduling. We next prove that *Adia*'s scheduling algorithm is 2-approximate in terms of link utilization when all access links have equal bandwidth.

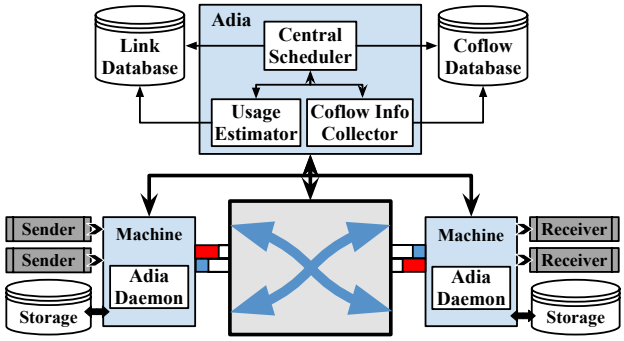


Fig. 3. The overview of *Adia*'s architecture: the central scheduler acquires up-to-date information about network states through the usage estimator and the information collector, and communicates with Adia daemons running on physical machines.

4 Design

In this section, we present the design of *Adia*, a hierarchical scheduling scheme to allocate bandwidth to flows of concurrent coflows. Since our primary objective is to maximize link utilization, we naturally choose to regard an access link as a scheduling unit when designing the online scheduling algorithms of *Adia*.

4.1 Framework Overview

To conduct coflow-aware scheduling, central monitoring is inevitable since no network devices or machines possess the information of all coflows. As we mentioned in Sec. 2, congestions only happen at access links. Therefore, *Adia* does not involve any functionalities or computations at switches. A daemon running on each physical machine is responsible for delivering coflow information and link status to the central scheduler. When a coflow's data is ready, the senders' daemons report the corresponding coflow information to the coflow info collector. When receivers are ready, the daemons estimate the states of the senders' uplinks and receivers' downlinks using existing techniques [28]. The central scheduler then determines the rate of each flow directly and informs the daemons to enforce endhost-based rate limiting for each flow. Nevertheless, such fine-grained scheduling at a single scheduler would be hard to scale out. As the number of coflows increases, the maintenance of flow states in each coflow is likely to slow down the schedule procedure.

To circumvent the overhead of a fully centralized scheduler, we turn to distribute the burden of fine-grained scheduling among *Adia*'s daemons across the network. The central scheduler is only responsible for maintaining global link states

and coflow information. Each daemon inquires the central scheduler periodically (or on-demand) to determine the rate of each flow running through it. Since the number of concurrent coflows are typically from tens to hundreds [12], the efficiency of the central scheduler is unlikely to become the performance bottleneck. Furthermore, the coflow size follows a heavy-tailed distribution [12]: about 98% of traffic is generated only by 8% of the coflows. By focusing on the large coflows, the side effect of the central coordination can be counteracted.

The framework of *Adia* is shown in Fig. 3. *Adia*'s daemons can be implemented in the application layer leveraging a client library to interact with data-parallel frameworks. We combine the objective of improving link utilization and coflow performance through a two-level hierarchical scheduling framework. At the first level, we try to maximize link utilization through reducing the longest completion times of all coflows. For flows that conflict at a common downlink, we differentiate among them based the uplink a flow comes from. For flows conflicting with each other at a common uplink, we further design a intra-link algorithm to minimize coflow completion times through priority-based scheduling. We schedule the flows on the same uplink based on the priority of the coflow it belongs to. The two algorithms reinforce each other since reducing coflow completion times essentially enables more concurrent coflows in the network. As a result, it calls for less time for all the coflows to finish.

4.2 Algorithms of Adia

In production data centers, coflows arrive to the system successively and we cannot predict the information of incoming coflows in advance. Therefore, the complicated combinatorial algorithms that work well in offline cases cannot be directly applied to schedule coflows in online cases. Instead, *Adia* only accounts for the currently active coflows, and is invoked whenever a new coflow arrives in, or an old coflow departs from the network.

As we have pointed out previously, minimizing the makespan is the key to maximize link utilization. Without virtual machine migration, the optimal makespan of any schedule equals to the net processing time of the heaviest loaded link [29]. It is worth noticing that the net processing time of a link is the time needed to send or receive data without any idle bandwidth. In other words, its value equals to the aggregated amount of traffic through a link divided by its bandwidth. Nevertheless, flows on the busiest link might be congested at the coupled links. Consequently, they cannot use up all the available bandwidth of the link, prolonging the makespan and reducing link utilization. Therefore, the key to minimizing the schedule makespan is to guarantee that the heaviest link experiences the least waiting time.

We achieve this design principle through the *largest load first* heuristic algorithm. The dynamic load of a link is defined as its net processing time as below

$$l_i^I \leftarrow \frac{\sum_j (d_{ij}^k - w_{ij}^k)}{B_i^I}, \quad l_i^E \leftarrow \frac{\sum_j (d_{ji}^k - w_{ji}^k)}{B_i^E} \tag{13}$$

w_{ij}^k indicates the amount of data f_{ij}^k has transmitted. The high-level scheduling then prioritizes uplinks with heavier loads when flows compete for downlink bandwidth. For flows on the same uplink, they may belong to different coflows. We try to minimize coflow completion times leveraging the *smallest remaining time first* algorithm. The completion time of a coflow is determined by its bottleneck flow that lags behind other flows in the same coflow. We estimate the completion time of each coflow if it exclusively occupies the network in line 9 of Algorithm 1, and use this value as its priority number. The low-level scheduling then conducts strict priority-based scheduling for flows belonging to different coflows on the same uplink. Flows belonging to the same coflow on a given uplink equally share the available bandwidth as in line 15. n_i^p and n_j^p are the number of flows on P_i^I and P_j^E that belongs to C_p respectively. The whole algorithm is shown in Algorithm 1 , which achieves the optimal schedule shown in Fig. 2(a) for our motivating example. Although *Adia* cannot guarantee the optimality under all circumstances, we show in Sec. 5 that its performance is much improved compared to existing schemes.

Algorithm 1 Online Scheduling Algorithm

1: **procedure** LARGESTLOADFIRST(M_k,W_k)
2: **for** $i = 1 : M$ **do**
3: $l_i^I \leftarrow \frac{1}{B_i^I} \sum_j \sum_k (d_{ij}^k - w_{ij}^k)$ ▷ current load on P_i^I
4: π = Sort($\{l_i^I\}$)
▷ Sort uplinks in the decreasing order of their loads
5: **return** π ▷ the permutation schedule of uplinks
6: **procedure** SMALLESTTIMEFIRST(B_R, i)
7: **Initiate:** $\mathcal{C} = \{C_k : \exists\, d_{ij}^k > 0\}$, $\Omega \leftarrow \mathcal{C}$
▷ the set of coflows on a given uplink
8: **for** $C_k \in \Omega$ **do**
9: $T_k \leftarrow \max(\max_i \frac{\sum_j s_{ij}^k - w_{ij}^k}{B_i^I}, \max_j \frac{\sum_i s_{ij}^k - w_{ij}^k}{B_j^E})$
10: **for** $p = 1 : |\mathcal{C}|$ **do**
11: $\pi(p) \leftarrow \arg\min_{C_k \in \Omega} T_k$ ▷ the shortest coflow
12: $\Omega \leftarrow \Omega \setminus \{C_{\pi(p)}\}$
13: **for** $p = \pi(1) : \pi(|\mathcal{C}|)$ **do**
14: **for** $j = 1 : m$ **do**
15: $r_{ij}^p = \min(\frac{B_R(P_i^I)}{n_i^p}, \frac{B_R(P_j^E)}{n_j^p})$
16: Update remaining bandwidth on P_i^I and P_j^E
17: **return** $\{r_{ij}^k(t) : C_k \in \mathcal{C}\}$ ▷ rates of flows on l_i^I
18: **procedure** MAIN
19: **Initiate:** $B_R \leftarrow \{B_i\}, W_k = \{0\}$
20: Π = LARGESTLOADFIRST(M_k,W_k)
21: **for** $i = \Pi(1) : \Pi(M)$ **do**
22: $\{r_{ij}^k(t)\}$= SMALLESTTIMEFIRST(B_R, i)
23: BACKFILLING(B_R, $\{r_{ij}^k\}$)
24: Update bandwidth B_R and transmitted data W_k

Algorithm 2 Backfilling Step

1: **procedure** BACKFILLING(B_R, $\{r_{ij}^k\}$)
2: **for** $i = 1 : m$ **do**
3: **while** $B_R(P_i^I) > 0$ **do**
4: **for all** k, j **do**
5: $\delta = \min(B_R(P_i^I), B_R(P_j^E))$
6: $r_{ij}^k(t) = r_{ij}^k(t) + \delta$
7: Update flow rates and idle bandwidth

4.3 Properties of Adia

We next demonstrate the scheduling algorithms in *Adia* achieve two important properties as stated below.

1) Work conserving: Through priority-based scheduling, bandwidth is first allocated to flows belonging to the fastest coflow on the busiest uplink. Other flows then utilize the remaining bandwidth according to their priorities. The rate allocating procedure finishes when all the links are saturated or no flow is suspended. We further integrate *backfilling* to make sure that all the available bandwidth is fully utilized by active coflows. In this way, we ensure the scheduling algorithm is work conserving.

2) Starvation free: Priority-based scheduling, however, usually suffers from starvation problem. Flows with lower priority may have to persistently wait for flows from higher priority classes to release bandwidth resources. *Adia* achieves starvation freedom by prioritizing links with larger loads and coflows with smaller completion times dynamically. At the high level, loads on low priority links will increase gradually, while loads on high priority links are likely to decrease. As a result, the priority of a link would increase relatively if its transmission has been throttled for a long time. Essentially, we adopt the *aging* mechanism to avoid starvation. At the low level, each uplink *softly reserves* a portion of bandwidth (denoted as α) for flows belonging to low priority coflows to proceed. Flows belonging to high priority coflows can only use at most $(1-\alpha)$ of uplink bandwidth if some flows on the same link are pending. The pending flows then fairly share the reserved bandwidth. By combine aging and multiplexing, we ensure no flow or coflow is perpetually suspended, and thus the scheduling mechanism is starvation free.

4.4 Analysis of Link Utilization

Scheduling jobs on a single machine is simple in terms of both link utilization and coflow performance since the resources are decoupled. We can maximize link utilization by keeping the machine working until all jobs complete, while we can achieve the optimal coflow performance by prioritizing the smallest coflows. It is NP-hard, however, to derive an optimal schedule across multiple access links [6] for either scheduling objective. The problem to maximize link utilization can be re-formulated as an open shop problem to minimize the makespan if all access links have the same capacity (see Appendix A). We regard an uplink as a job to be scheduled, and a downlink as a working machine. The flows on a given uplink then correspond to constituent operations of the job, to be scheduled onto different machines. In this way, the interdependency of uplink and downlink capacities is transformed to the concurrency constraints in the open shop scheduling problem. We next prove that our largest load first algorithm for inter-link scheduling is 2-approximate for offline cases.

Since all links are identical, we suppose the bandwidth of all links is one unit without loss of generality. A feasible schedule S can be depicted as

$$S = \{[b_{ij}, e_{ij}); \forall i, j, k\} \tag{14}$$

where b_{ij} and e_{ij} are the beginning and ending times of the transmissions from uplink i to down link j. We further introduce two functions as in [30]:

$$\text{Machine idle: } R_j(t) = \begin{cases} 1, & t \notin \bigcup_{i=1}^{M} [b_{ij}, e_{ij}) \\ 0, & t \in \bigcup_{i=1}^{M} [b_{ij}, e_{ij}) \end{cases} \tag{15}$$

$$\text{Job in-progress: } Q_i(t) = \begin{cases} 1, & t \in \bigcup_{j-1}^{M} [b_{ij}, e_{ij}) \\ 0, & t \notin \bigcup_{j=1}^{M} [b_{ij}, e_{ij}) \end{cases} \tag{16}$$

S is called dense if and only if $R_j(t) = 1$ implies $Q_i(t) = 1$ for all i such that $b_{ij} > t$. We now prove that the our scheduling algorithm derives an open-shop dense schedule.

As uplinks and downlinks all have unit capacity, the transmission from an uplink to a downlink is either at unit rate or suspended. According to the inter-link scheduling algorithm, we select uplinks to start transmission in the decreasing order of their pending loads. Uplinks with low priorities then select from transmission to downlinks that have not been occupied by high priority uplinks (*i.e.*, selecting downlinks with $R_j(t) = 0$). After all uplinks have scheduled to start transmission, a job (uplink) is suspended only when all the machines corresponding to its unscheduled transmissions are busy. Consequently, we have $Q_i(t) = 0$ implies $R_j(t) = 0$ for $\forall\ j \in \{j : b_{ij} > t\}$. Correspondingly, $R_j(t) = 1$ implies $Q_i(t) = 1$ for $\forall\ i \in \{i : b_{ij} > t\}$. This completes the proof that our algorithm achieves a dense scheduling in offline cases. Since the worst-case makespan of any dense scheduling is no more than twice of the optimal makespan [30], our high level scheduling algorithm is 2-approximate in terms of link utilization.

4.5 Analysis of Coflow Completion Time

The shortest remaining time first algorithm optimize coflow performance if all coflows are on a single link [27]. As for the concurrent open shop scheduling problem, a primal-dual 2-approximate algorithm is proposed by Mastrolilli *et al.* in [14]. This offline algorithm cannot be directly applied to our online scenario. As analyzed in Sec. 3, the scheduling problem to improve coflow performance is a variant of concurrent open shop scheduling problem. Even if all access links have equal capacities, machines (uplinks and downlinks) are still interdependent. Therefore, it is theoretically difficult to find a scheduling algorithm with performance bound.

We try to improve the coflow performance by approximating the shortest remaining time first strategy at the low level of *Adia*. Through prioritizing flows belonging to *faster* coflows, we improve coflow performance in comparison with flow-level scheduling schemes. In practical data center networks, access links are heterogeneous: the capacities of uplinks and downlinks vary significantly across the network. It is hard to theoretically analyze the performance of *Adia* due to the coupled resources of uplinks and downlinks. We turn to evaluate *Adia* in practical networks through extensive simulations.

5 Experimental Evaluation

In this section, we evaluate the performance of *Adia* through extensive packet-level simulations in the ns3 [15] simulator. The implementation can be found at this link. We compare *Adia* with existing flow-level and coflow-level schemes in terms of both link utilization and average coflow completion time (CCT), and examine their performance under varying network loads. We further evaluate the influence of limited multiplexing by varying the fraction of reserved bandwidth.

5.1 Simulation Methodology

5.1.1 Benchmark workloads

We use empirical workloads to reflect coflow patterns observed in production data centers [12]. All access links have

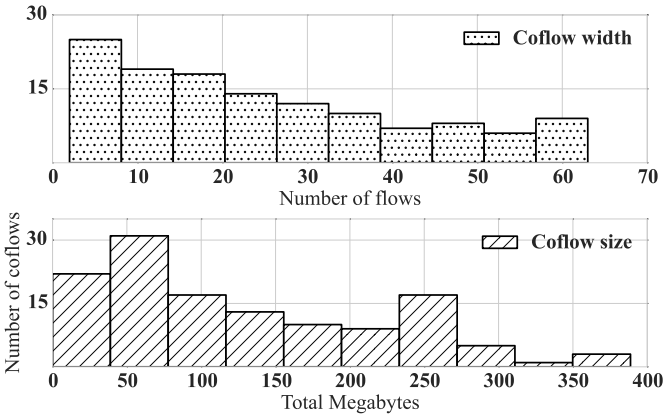


Fig. 4. The histograms of coflow sizes and widths: the figure on the top illustrates that about 80% of coflows consist of less than 40 flows; the figure on the bottom shows that coflow sizes follows a long-tail distribution with less than 20% of coflows contribute to more than 80% of network traffic.

1 Gbps bandwidth. We capture the characteristics of a coflow through its width (the number of constituent flows) and size (the overall amount of data). Practical traces have shown that coflow sizes follow a long-tail distribution: only about 18% of coflows contribute to more than 80% of network traffic. In addition, about 78% of coflows have less than 40 flows. The histograms of coflow sizes and widths are shown in Fig. 4. Coflows arrive to the network according to a Poisson process with rate $\lambda \in [0.2, 0.8]$ to reflect varying network loads. Since we focus on bandwidth allocation on access links, how to place senders and receivers is irrelevant. We evenly place senders and receivers across machines in a round-robin fashion. The ratio between the number of senders and receivers varies across different coflows. But the number of senders is larger than the number of receivers in about 80% coflows.

5.1.2 Schemes compared

We analyze the performance of *Adia* by comparing it with the following scheduling mechanisms.

Per-flow fairness: Per-flow fairness strategy is widely adopted in state-of-the-art transport protocols, such as TCP and its data center variant, DCTCP [31]. When flows contend for bandwidth on a congested link, they equally share the bandwidth on that link to achieve the max-min fairness. DCTCP focuses on minimizing queuing delay at switches through ECN-based congestion control algorithms. For large flows which tend to be the slowest in a coflow, the queuing delay only accounts for a negligible fraction of completion time [16]. We expect the coflow-level performance of TCP and DCTCP is similar and only make comparisons with TCP.

Per-flow priority: Since fair sharing strategies hurt flow performance as demonstrated in [5], [6], recent flow scheduling schemes propose to adopt priority-based scheduling. Such a strategy embraces preemptive scheduling and assign each flow a single priority number based on different information, such as the remaining size in pFabric [6] and the remaining time to its deadline in PDQ [5]. We compare *Adia* with pFabric and omit the comparison with PDQ due to its different objective. Since the core networks are congestion-free, the scheduling algorithm can get rid of the network inefficiencies caused by packet drops, load imbalance and queuing delay. Essentially, we implement the ideal algorithm proposed in pFabric, whose performance is better than pFabric.

Coflow scheduling: The essence of coflow scheduling lies in the per-coflow priority based scheduling strategy, such as the smallest bottleneck first strategy used in Varys [12] and the least attained service first strategy used in Aalo [24]. Since Aalo achieves similar performance with Varys, we use Varys as the representative since it also utilizes the prior information as *Adia* does. The dynamic routing used in Rapier [23] is irrelevant since the core network is congestion-free.

5.1.3 Performance metrics

Average link utilization: Unlike the offline case we have discussed in Sec. 3, the coflows do not arrive in the network at the same time in our online simulations. Therefore, some ports may not have any traffic demand in the beginning and thus remain idle regardless of the scheduling schemes. Therefore, we record the utilization of each link throughout its own active time, which is defined as the period from the time point it starts transmitting data till all the data on that link are sent. This is different from the definition of in the offline case, but we are able to analyze fine-grained link utilization states and locate the bottlenecks in the network.

Average CCT: We also collect the information of each coflow's completion time to evaluate the average coflow completion time, which is the most commonly used metric to evaluate coflow-level performance [12], [24].

5.2 *Adia*'s Performance

From the results in Fig. 5, we can see that *Adia* significantly improves the average link utilization: compared to Varys, pFabric and TCP, *Adia* improves the average link utilization by about 19.04%, 16.82% and 22.95% ($\lambda = 0.2$). It is worth noticing that Varys shows no advantage over per-flow schemes in terms of link utilization since it only guarantees work conservation without considering better utilization. As for the coflow performance (Fig. 6), due to embracing coflow semantics, *Adia* is able to speed up coflows by about $2\times$, which is comparable with Varys.

Since network loads significantly influence link utilization, we evaluate the reaction to different network loads through varying the intensity (λ) of coflows' arrival process. Under a low network load, the interval of coflow arrivals is large, and thus the number of concurrent coflows in the network is small. Links might be left underutilized or even idle. When the network is heavily loaded, the links are more likely to be saturated. As for the coflow completion time, the competition intensity is severer when more concurrent coflows run simultaneously. As a result, coflows get less bandwidth to transmit data, and their completion times would increase accordingly.

We vary coflows' arrival rate, λ, from 0.2 to 0.8 to examine the performance under different network loads. In Fig. 5(a) and Fig. 6(a), the link utilizations and the coflow completion times of the four schemes increase as the network becomes busier. *Adia* significantly outperforms other schemes under different network loads. pFabric achieves higher utilization than traditional TCP as it can effectively reduce the average flow completion time and improve network efficiency. Similarly, Varys achieves similar link utilization with pFabric since it make all the flows inside a coflow finish at the same time to admit more coflows into the network. With respect coflow performance, Varys and *Adia* significantly outperform per-flow mechanisms. Furthermore, *Adia* further reduces coflow

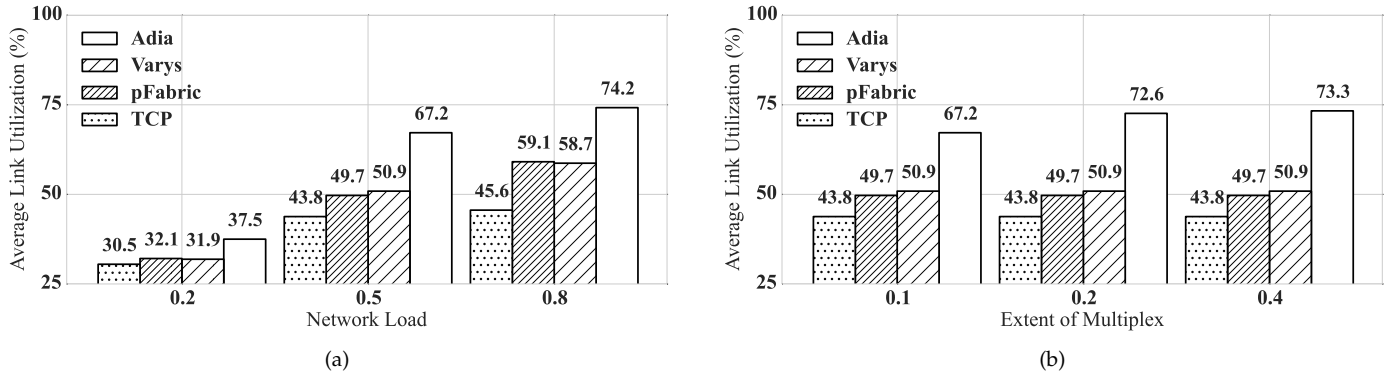


Fig. 5. The link utilizations of four schemes under varying network loads ((a) $\alpha = 0.1$.) and varying extents of multiplexing ((b) $\lambda = 0.5$).

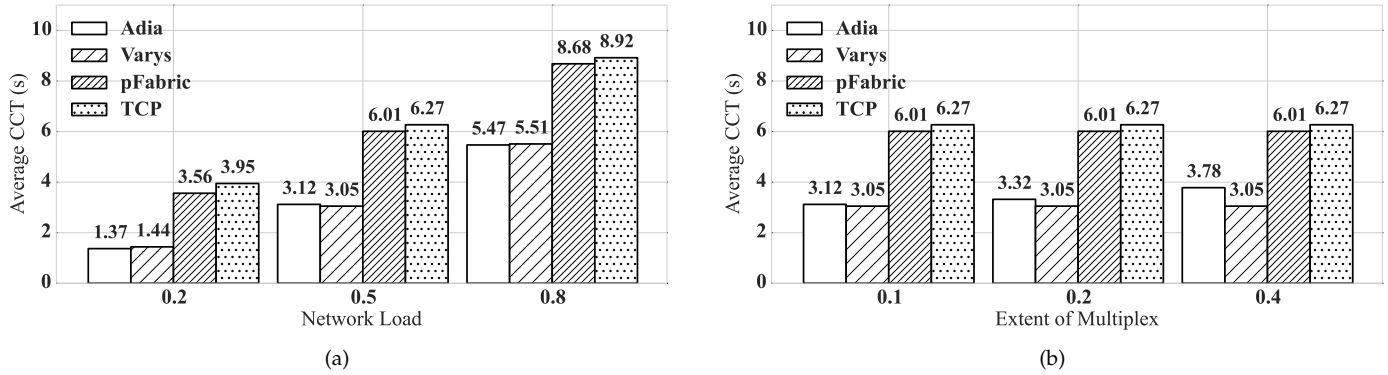


Fig. 6. The coflow completion times of four schemes under varying network loads ((a) $\alpha = 0.1$.) and varying extents of multiplexing ((b) $\lambda = 0.5$).

completion times when network load is high due to its ability in fully utilizing link bandwidth.

To avoid starvation, *Adia* has introduced limited multiplexing to let low priority coflows share the reserved bandwidth. It is clear that the extent of multiplexing influences both link utilization and coflow performance. We change the portion of reserved bandwidth (α) in the group of experiments shown in Fig. 5(b) and 6(b). As the network load and other parameters remain the same, the performance benchmark schemes is not influenced. In contrast, the link utilization under *Adia* increases with the increase of reserved bandwidth.

However, with more bandwidth reserved for multiplexed coflows, it is more likely for flows to fairly share the reserved bandwidth. In the extreme case, *Adia* would fall back to the per-flow fairness scheme. As a result, the coflow completion times are prolonged. It is also worth noticing that the increase of link utilization decreases with a larger extent of multiplexing. In our simulations, we observe that the link utilization would start to decrease when α is larger than 0.4. For the best practice, we could find the extend of multiplexing that achieves the highest link utilization. The best value of α depends on the actual workloads and should be tuned dynamically. In general, we suggest to choose a small α to guarantee the coflow performance.

6 Related Work

Researchers have made continuously efforts to maximize network throughput and improve link utilization. McKeown in [32] proposes iSLIP, a scheduling algorithm for input queues of crossbar switches to achieve 100% throughput when the traffic is uniform. Recent flow-level scheduling algorithms (*e.g.*, [6], [4], [5]) struggle to achieve work conservation. As we have argued previously, work conservation is insufficient to achieve the optimal link efficiency. SWAN [1] and B4 [22] further propose to leverage software-defined networking to maximize the utilization of inter-datacenter links. By dividing the traffic into coarse-grained classes, the scheduler distinguishes flows with different urgencies. Although the two mechanisms effectively improve network utilization, but similar to other flow-level schemes, neither of them considers coflow performance. As a result, bandwidth might be wasted to flows that already finish ahead of the bottleneck of the coflow. This potentially slows down the bottleneck flow and the overall coflow performance. We contend that maximizing link utilization should take coflow performance into account at the same time.

Existing coflow scheduling schemes, however, have merely focused on the average completion times of coflows. Chowdhury *et al.* in [11] propose to centrally determine the maximum number of TCP connections a coflow can set up, and approximate both inter- and intra- coflow weighted fair sharing. Such a pure multiplex strategy cannot improve coflow performance as pointed out in recent proposals [13], [12], which turn to leverage priority-based scheduling strategies.

Baraat [13] adopts FIFO with limited multiplexing to reduce the average coflow completion time. In contrast, we conduct two-level scheduling to minimize the makespan and coflow completion time at the same time. Varys [12] provides an intent-driven API for data-parallel coflows to convey their information to a central scheduler, which performs the smallest bottleneck first heuristic to minimize average completion times. Aalo [24] further simplifies the coflow scheduling with-

out the need to acquire coflow information in advance. Qiu *et al.* in [33] theoretically analyze the problem of minimizing the weighted coflow completion times and propose algorithms that have approximation bounds. Since the schemes mentioned above all center on the coflow completion time, and struggle to guarantee work conservation, the essence of achieving high link utilization has not been fully explored. Another recent work, HUG [25], achieves high link utilization and considers the correlated demands of applications (coflows) at the same time. However, it sacrifices utilization for strategy proofness in public clouds.

Apart from bandwidth allocation and scheduling, Alizadeh *et al.* propose Conga [34], a in-network load balancing mechanism, to detect global congestion information and distribute network loads in a balanced manner. The network throughput and thus link utilization can be effectively improved. Conga, however, requires upgrade of existing switches in data centers and is unaware of coflow performance. In contrast, *Adia* is a hypervisor-based scheme and can be built upon their improved hardware silicons to further improves link utilization and coflow performance at the same time. Zhao *et al.* in [23] try to reduce the completion time of a coflow by combine coflow scheduling and dynamic routing via Rapier. Leveraging OpenFlow-enabled switches, flows are routed to a centrally computed path, rather than following the default ECMP routing. Nevertheless, leveraging dynamic routing restricts their method to software-defined data centers. In addition, the control overhead for a central controller to deploy routing rules potentially harms coflow performance.

7 Conclusion

In this paper, we have proposed and studied the problem of maximizing link utilization with coflow-aware scheduling, which is NP-hard in offline scenarios. We have designed and implemented a hierarchical online scheduling mechanism, *Adia*, to conduct both inter- and intra- link scheduling. Specifically, for inter-link scheduling, we compute the load of each uplink, and prioritize flows on an uplink with larger processing time; for intra-link scheduling, we try to minimize coflow completion times leveraging the shortest remaining time first strategy. *Adia* is both work-conserving and starvation-free, making it practical for online application. Through theoretic analyses, we prove the simple yet effective heuristic algorithm in *Adia* is 2-approximate when access links are homogeneous. We demonstrate that *Adia* is able to maximize link utilization without loss of coflow performance through extensive realistic simulations.

Appendix A
Complexity Analysis

The NP-hardness of minimizing CCT is proved in [12] by means of reducing the NP-hard *concurrent* open shop scheduling problem [27] to it. We next prove the hardness of the coflow scheduling problem to maximize link utilization as below.

Theorem 1. *Optimizing L in the offline case is NP-hard for all $m > 2$ even when all access links have uniform capacities.*

Proof. Given an open shop scheduling instance with m independent machines and n jobs, we construct a coflow scheduling instance where each downlink corresponds to a machine, and each uplink corresponds to a job. All the downlinks and uplinks have unit (indivisible) capacity . The problem to schedule jobs on multiple machines is then transformed to schedule flows on each uplink to multiple downlinks with the objective of minimizing the schedule makespan.

Each job in the open shop scheduling instance consists of multiple operations, each of which needs to be processed on a specific machine for a given period of time. Accordingly, each uplink in our coflow scheduling instance consists of multiple flows, each of which needs to send a corresponding amount of data to multiple downlinks. The concurrency restrictions in the open shop instance require that: each job can be processed only at one machine at a time, and each machine can process at most one job at a time. The first part of the requirements transform to the capacity constraints in Eq. (8), which require each uplink (job) can at most send data to one downlink (machine) at unit rate. The second part of the requirements transform to the capacity constraints in Eq. (9), which require each downlink (machine) can at most receive data from one uplink (job) at unit rate. Therefore, we successfully reduce an arbitrary open shop scheduling instance to a coflow scheduling problem in a network where all uplinks and downlinks have unit capacity.

Since the open shop scheduling problem with the objective of minimizing the schedule makespan is NP-hard [26], our scheduling problem formulated in Eq. (7)-Eq. (9) is NP-hard as well. □

When link capacities are non-uniform, the concurrency constraints need to be generalized. For an ingress port P_i^I, denote the set of egress ports it can simultaneously transmit data to as G_i. The capacities of egress ports in G_i satisfy:

$$\sum_{P_j^E \in G_i} B_j^E \leq B_i^I$$

The same rule also applies to any egress port. It is clear to see that the level of concurrency ($|G_i|$) actually depends on the destinations of flows that currently run through P_i^I. In other words, the concurrency constraints dynamically change over the time and are heterogeneous across the network. Therefore, our scheduling problem is much more complicated than the open shop scheduling problem in practice.

References

[1] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, 2013.

[2] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2008.

[3] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM conference on Internet measurement (IMC)*, 2010, pp. 267–280.

[4] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 50–61, 2011.

[5] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 127–138, 2012.

[6] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM*, 2013.

[7] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *Proc ACM SIGCOMM*, 2012.

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proc. USENIX OSDI*, 2004.

[9] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 59–72, 2007.

[10] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proc. ACM SIGMOD*, 2010.

[11] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 98–109, 2011.

[12] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient Coflow Scheduling with Varys," in *Proc. ACM SIGCOMM*, 2014, pp. 443–454.

[13] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. ACM SIGCOMM*, 2014.

[14] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan, "Minimizing the sum of weighted completion times in a concurrent open shop," *Operations Research Letters*, vol. 38, no. 5, pp. 390–395, 2010.

[15] "The Network Simulator NS-3." http://www.nsnam.org/.

[16] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: Practical network performance isolation at the edge," in *Proc. USENIX NSDI*, 2013.

[17] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[18] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, 2009.

[19] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 75–86, 2008.

[20] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly." in *Proc. USENIX NSDI*, 2012.

[21] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou, "Re-optimizing data-parallel computing," in *Proc. USENIX NSDI*, 2012.

[22] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM*, 2013.

[23] Y. Zhao, K. Chen, W. Bai, M. Y. USC, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. IEEE INFOCOM*, 2015.

[24] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. ACM SIGCOMM*, 2015, pp. 393–406.

[25] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "HUG: Multi-resource fairness for correlated and elastic demands," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016, pp. 407–424.

[26] D. Bai and L. Tang, "Open shop scheduling problem to minimize makespan with release dates," *Applied Mathematical Modelling*, vol. 37, no. 4, pp. 2008 – 2015, 2013.

[27] T. A. Roemer, "A note on the complexity of the concurrent open shop problem," *Journal of scheduling*, vol. 9, no. 4, pp. 389–396, 2006.

[28] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, 2013, pp. 231–242.

[29] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *Journal of the ACM (JACM)*, vol. 23, no. 4, pp. 665–679, 1976.

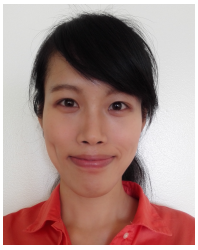
[30] R. Chen, W. Huang, Z. Men, and G. Tang, "Open-shop dense schedules: properties and worst-case performance ratio," *Journal of Scheduling*, vol. 15, no. 1, pp. 3–11, 2012.

[31] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 63–74, 2011.

[32] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Trans. Communications*, vol. 47, no. 8, pp. 1260–1267, 1999.

[33] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the total weighted completion time of coflows in datacenter networks," in *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2015, pp. 294–303.

[34] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 503–514.



Jingjie Jiang received the B.Eng. degree from the Department of Automation, Tsinghua University, China, in 2012. Since 2012, she has been with the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology, where she is currently a PhD candidate. She visited the Department of Electrical and Computer Engineering at the University of Toronto during March to August in 2015. She is a member of IEEE. Her current research interests include: datacenter networking, resource allocation and job scheduling.

Shiyao Ma received his B.Eng. degree in the Computer Science, Tsinghua University, Beijing, in 2013. Since then, he has been with the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology, where he is currently a PhD candidate. He was a visiting student in the Department of Electrical and Computer Engineering at the University of Toronto from March 2016 to August 2016. His current research interests include datacenter networking and job scheduling.

Bo Li is a professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He is a Fellow of IEEE. He was the Chief Technical Advisor for ChinaCache Corp.(NASDAQ CCIH), the largest CDN operator in China. He was a Cheung Kong Visiting Chair Professor in Shanghai Jiao Tong University (2010-2013) and an adjunct researcher in Microsoft Research Asia (1999-2007) and in Microsoft Advance Technology Center (2007-2009). His current research interests include: multimedia communications, the Internet content distribution, datacenter networking, cloud computing, and wireless sensor networks.

He made pioneering contributions in the Internet video broadcast with the system, Coolstreaming, which was credited as the world first large-scale Peer-to-Peer live video streaming system. The work appeared in IEEE INFOCOM (2005) received the IEEE INFOCOM 2015 Test-of-Time Award. He has been an editor or a guest editor for over a dozen of IEEE journals and magazines. He was the Co-TPC Chair for IEEE INFOCOM 2004.

He received five Best Paper Awards from IEEE. He received the Young Investigator Award from Natural Science Foundation of China (NFSC) in 2005, the State Natural Science Award (2nd Class) from China in 2011. He received his B. Eng. in the Computer Science from Tsinghua University, Beijing, and his PhD in the Electrical and Computer Engineering from University of Massachusetts at Amherst.

Baochun Li received the B.Engr. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000.

Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks.

Dr. Li has co-authored more than 290 research papers, with a total of over 13000 citations, an H-index of 59 and an i10-index of 189, according to Google Scholar Citations. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM and a Fellow of IEEE.