Maximizing Link Utilization with Coflow-Aware Scheduling in Datacenter Networks

Jingjie Jiang*, Shiyao Ma*, Bo Li*, Baochun Li†, and Jiangchuan Liu ‡
*Department of Computer Science and Engineering, Hong Kong University of Science and Technology
†Department of Electrical and Computer Engineering, University of Toronto
‡South China Agricultural University

Abstract—**Link utilization has received extensive attention since datacenters become the most prevalent platform for data-parallel computing applications. A specific job of such applications involves communication among multiple machines. The *coflow* abstraction depicts such communication and captures application performance through corresponding network requirements. Existing techniques to improve link utilization, however, either restrict themselves to 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*. Despite the hardness of this problem, we design *Maluca*, a hierarchical scheduling framework to conduct both inter- and intra- link scheduling. *Maluca*'s algorithm is not only starvation-free and work-conserving, but also 2-approximate in terms of link utilization. Extensive simulation results demonstrate that *Maluca* outperforms both per-flow and coflow schemes in terms of link utilization, and achieves similar coflow performance in comparison with the state-of-art coflow scheduling schemes.**

I. Introduction

Achieving high link utilization has come into the academic spotlight since datacenters become the *de facto* computing platform for data-parallel applications [1]. Although the network links are very costly to deploy, the *average* utilization of network links is unfortunately very low (Less than 40%) in most datacenters [1] [2]. 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.

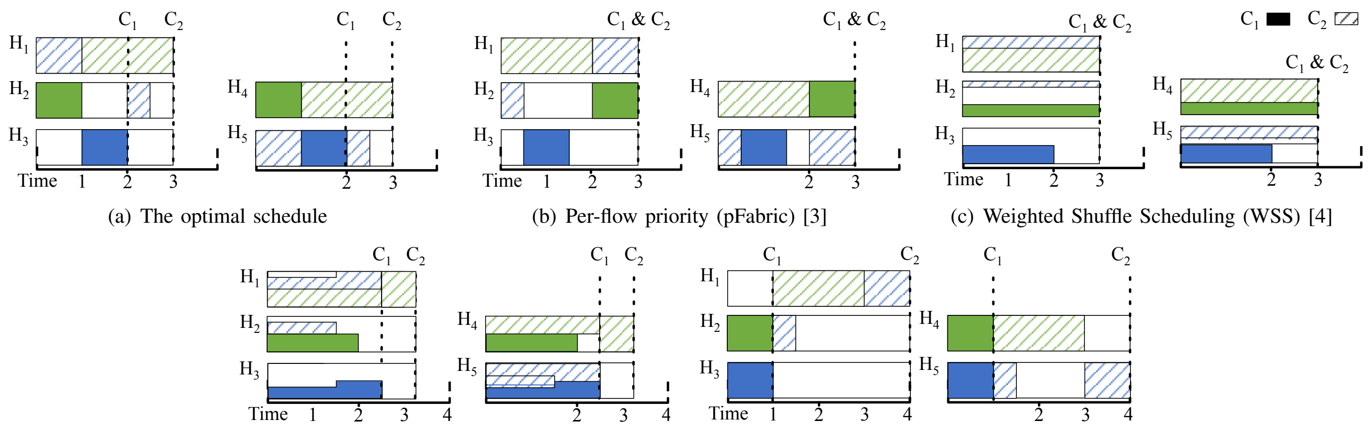
To improve link utilization, existing flow scheduling mechanisms strive to be work-conserving [3] . Work conservation, however, only tries to improve the utilization of each single link without considering the global situation. Without proper coordination, a flow might unnecessarily occupy the bandwidth of a receiver's downlink, hindering other flows on the same link. Consequently, the uplinks of these affected flows are at risks of underutilization. In such cases, work conservation is not violated, but the link utilization is suboptimal. Mechanisms with work conservation are thus insufficient to achieve optimal link utilization. To make things worse, datacenter tenants may hide their true traffic demands and throttle some of their data transfers on purpose to acquire more bandwidth on a congested link. The datacenter operator would then deem that the network is work conserving, while the actual link utilization is very low.

The research was support in part by grants from RGC under the contracts 615613, 16211715 and C7036-15G (CRF), a grant from NSF (China) under the contract U1301253.

Since data-parallel jobs are usually network-bounded [4], the response times depend on network transfers within each job. Such job-specific communication involves multiple parallel flows to transmit data among groups of machines in successive computation stages [4]. The *coflow* abstraction [5] formally defines a group of concurrent flows that transfer intermediate results among machines as a coflow. Until the completion of *all* the constituent flows, will a coflow be considered completed. Given such correlation of flows, even if flow-level scheduling 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*, 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 access links. *Secondly*, high link utilization must be achieved without sacrificing coflow performance. As demonstrated previously [5], coflow-level priority scheduling effectively reduces the average coflow completion time. Nevertheless, their coflow-centric scheduling hierarchy may conflict with our link-oriented 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 wisdom in operation research and combinatorics [6], which need information about all the coflows in advance.

We propose to address these three challenges by designing an online framework, *Maluca*, to maximize link utilization with coflow-awareness. Based on an in-depth analysis of the coflow scheduling problem, we prove maximizing link utilization is NP-hard by reducing the *open shop scheduling* problem to it. To handle the correlation between link utilization and coflow performance, we design a *hierarchical* scheduling mechanism to conduct both inter- and intra- link scheduling. At the first level, *Maluca* treats one link as an entity, and conducts inter-link scheduling to reduce the makespan of the schedule. At the second level, *Maluca* zooms in to flows belonging to the same link, and performs intra-link scheduling to minimize the average coflow completion time. As reducing coflow completion times can improve system throughput, intra-link scheduling essentially reinforces the performance of inter-link scheduling. We further prove our algorithm is 2-approximate when all access links are identical. We evaluate the performance of *Maluca* using *ns3*. The extensive simulation results verify that *Maluca* improves link utilization, and achieves similar coflow performance in comparison with [5], the state-of-art coflow scheduling scheme.



(a) The optimal schedule (b) Per-flow priority (pFabric) [3] (c) Weighted Shuffle Scheduling (WSS) [4]

(d) TCP fair sharing (e) Smallest Effective Bottleneck First (SEBF) [5], Minimum Completion Time First [7]

Fig. 1. 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.

II. Motivation and Background

A. System Model

According to the statistics collected in production datacenters [8], core networks seldom experience severe or persistent congestion, while network edges are often congested. Given such observations, we suppose congestions only occur at network edges for simplicity. The whole datacenter fabric can then be viewed as a non-blocking switch. Under such a network model, the only scarce network resource is the bandwidth of access links. have no influence on scheduling strategies.

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 an idle link can be offline to save power and thus the idle time does not influence link utilization. However, 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. For instance, in Fig. 1(a), H_2 is idle at time 1 but it needs to transfer data at time 2. Without priori information about when coflows arrive, turning down idle links will either harm network availability of incur too frequent on-offs and consume even more power. Therefore, we contend that idle links have to remain online as in [1]. Possible energy saving schemes are beyond the scope of this paper.

B. Motivating Example

Expediting flows or coflows as in [3], [4], [5], [9] 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. 1. Three hosts, H_1, H_2 and H_3, initiate five flows to the two receivers, H_4 and H_5, in the network where access links have unit capacity.

It is easy to verify that all the five schedules are work-conserving, but the link utilization and coflow performance vary significantly. The per-flow fair sharing strategy equally shares the bandwidth among all the flows, and achieves the results in Fig. 1(d): the two coflows finish within 2.5 and 3.25 time units, and the average link utilization is 67.7%. pFabric [3] (Fig. 1(b)) prioritizes smaller flows, and makes the two coflows finish at 3 time units with the link utilization equal to 73.3%. Orchestra [4] aims to reduce coflow completion times but only achieves the same performance as pFabric. In contrast, the smallest effective bottleneck first algorithm in Varys [5] and the minimum completion time first strategy in Rapier [7] 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. 1(a) achieves the same average coflow completion time but raises the average link utilization to 73.3%.

Through this example, we demonstrate that achieving work conservation is *necessary yet insufficient* to maximize link utilization. Furthermore, the makespan of all coflows, rather than their average completion time, determines link utilization. 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.

III. Problem Formulation

Given a set of coflows running in datacenters, 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 datacenter 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 [5], [9], 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. Replacing 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 $L = \max_k T_k$ denote the makespan (or the length) of a schedule, which indicates that all the coflows have completed transmission after L time units. Therefore, we can derive that

$$s_{ij} = \sum_k s_{ij}^k = \int_0^L r_{ij}(t)\, dt, \ \forall i, \ \forall j \tag{4}$$

$$\text{where} \quad r_{ij}(t) = \sum_k r_{ij}^k(t) \tag{5}$$

To find a feasible schedule, any access link should not be overloaded. 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. (7) and (8). Together with the constraint (3), the problem of maximizing the aggregated utilization of all access links can be formulated as

$$\max \quad \frac{1}{L} \int_0^L \frac{\sum_i \sum_j (r_{ij}(t) + r_{ji}(t))}{\sum_i (B_i^I + B_i^E)}\, dt \tag{6}$$

$$\text{s.t.} \quad \sum_j r_{ij}(t) \le B_i^I, \qquad \forall t, \ \forall i \tag{7}$$

$$\sum_j r_{ji}(t) \le B_i^E, \qquad \forall t, \ \forall i \tag{8}$$

Since the egress and ingress bandwidths add up to a port's total bandwidth, Eq. (6) can be transformed to

$$\frac{1}{L} \frac{\sum_i \sum_j \int_0^L (r_{ji}(t) + r_{ij}(t))\, dt}{\sum_i B_i} \tag{9}$$

Replacing the integral term with s_{ij} as in Eq. (4) yields

$$\frac{1}{L} \frac{\sum_i \sum_j (s_{ji} + s_{ij})}{\sum_i B_i} \tag{10}$$

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 (10) as a variant of *open shop scheduling problem* [10]. The dependencies among access links can then be transformed to *heterogeneous concurrency* constraints. 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):

$$\min \quad CCT = \frac{1}{n} \sum_{k=1}^{n} T_k \tag{11}$$

$$\text{s.t.} \quad (3)\ (7)\ (8) \ \text{ hold}$$

Minimizing the makespan or average coflow completion time is NP-hard even when all links are homogeneous and all coflows start at the same time with prior knowledge. As a sketch of proof, the NP-complete *open shop scheduling problem* [10] can be reduced to our problem to minimize the makespan; the NP-hard concurrent open shop scheduling problem can be reduced to the problem to minimize average coflow completion time. It is not always possible to find an optimal schedule that both minimizes the average completion time and the makespan. We need to make a trade-off between the coflow performance and link utilization.

IV. Design

A. Framework Overview

The framework of *Maluca* is shown in Fig. 2. *Maluca* 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 information to the coflow info collector. When receivers are ready, the daemons estimate the states of the senders' uplinks and receivers' downlinks. 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, each daemon inquires the central scheduler periodically (or on-demand) to acquire the link and coflow information and determine the rate of each flow running through it. Since the number of concurrent coflows are typically from tens to hundreds [5], the efficiency of the central scheduler is unlikely to become the performance bottleneck. Furthermore, the coflow size follows a heavy-tailed distribution [5]: 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.

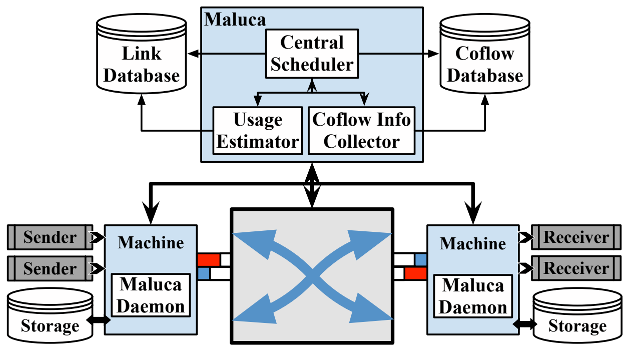


Fig. 2. *Maluca*'s architecture: the central scheduler acquires up-to-date information about network states through the usage estimator and the information collector, and communicates with *Maluca* daemons running on physical machines.

B. Algorithms of *Maluca*

In production datacenters, 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, *Maluca* 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 proved 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. The net processing time of a link is 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, and will prolong the makespan and harm link utilization. Therefore, the key to minimizing the schedule makespan is to guarantee that the heaviest link experiences the least waiting time.

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:      $\pi$ = Sort($\{l_i^I\}$)
                ▷ Sort uplinks in the decreasing order of their loads
5:      return $\pi$          ▷ 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 BackFilling($B_R$, $\{r_{ij}^k\}$)
19:     for $i = 1 : m$ do
20:         while $B_R(P_i^I) > 0$ do
21:             for all $k, j$ do
22:                 $\delta = \min(B_R(P_i^I), B_R(P_j^E))$
23:                 $r_{ij}^k(t) = r_{ij}^k(t) + \delta$
24:                 Update flow rates and idle bandwidth
25: procedure Main
26:     Initiate: $B_R \leftarrow \{B_i\}, W_k = \{0\}$
27:     $\Pi$ = LargestLoadFirst($M_k$,$W_k$)
28:     for $i = \Pi(1) : \Pi(M)$ do
29:         $\{r_{ij}^k(t)\}$= SmallestTimeFirst($B_R$, $i$)
30:     BackFilling($B_R$, $\{r_{ij}^k\}$)
31:     Update bandwidth $B_R$ and transmitted data $W_k$
```

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

$$l_i^I \leftarrow \frac{\sum_j (d_{ij}^k - w_{ij}^k)}{B_i^I}, \quad l_i^E \leftarrow \frac{\sum_j (d_{ji}^k - w_{ji}^k)}{B_i^E} \tag{12}$$

w_{ij}^k indicates the amount of data f_{ij}^k has transmitted. *Maluca* then prioritizes uplinks with heavier loads when competing for downlink bandwidth. For flows on the same uplink, we try to improve coflow performance via the *smallest remaining time first* algorithm. 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 link. Flows belonging to the same coflow on a given link 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. Although *Maluca* cannot guarantee the optimality under all circumstances, we show in Sec. V that its performance is much improved compared to existing schemes.

C. Properties of *Maluca*

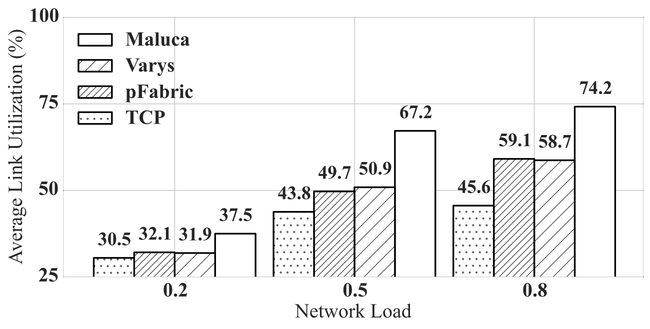
We next demonstrate the scheduling algorithms in *Maluca* achieve two important properties as stated below.

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.

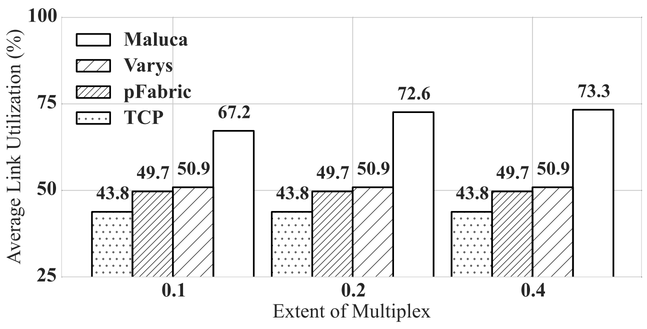
2) Starvation free: Priority-based scheduling usually suffers from starvation problem. *Maluca* achieves starvation freedom by prioritizing links with larger loads dynamically. Loads on low priority links will increase gradually, while loads on high priority links are likely to decrease. As a result, 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. Each uplink *softly reserves* a portion of bandwidth (denoted as α) for flows belonging to low priority coflows to proceed. By combine aging and multiplexing, we ensure no flow or coflow is perpetually suspended, and thus the scheduling mechanism is starvation free.

3) High link utilization: Scheduling jobs on a single machine is simple 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 [11]. It is NP-hard, however, to derive an optimal schedule across multiple access links [3] for either scheduling objective. We can prove that our largest load first algorithm for inter-link scheduling is 2-approximate for offline cases since it is a dense schedule [12]. We skip the detailed proof due to the limited space.

4) Improved coflow performance: Due to the coupled resources of uplinks and downlinks, it is theoretically difficult to find a scheduling algorithm with performance bound in terms of coflow completion time. We turn to evaluate *Maluca* in practical networks through extensive simulations.



(a)



(b)

Fig. 3. The link utilizations of four schemes under varying network loads ((a) $\alpha = 0.1$.) and varying extents of multiplexing ((b) $\lambda = 0.5$).

V. Experimental Evaluation

A. Simulation Methodology

We use empirical workloads to reflect coflow patterns observed in production datacenters [5]. All access links have 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. 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.

We analyze the performance of *Maluca* by comparing it with the three types of scheduling mechanisms: per-flow fairness (*i.e.*, TCP), per-flow priority (pFabric [3]) and coflow scheduling (Varys [5]). Our evaluations and comparisons use two commonly-used performance metrics: *average link utilization* and *average CCT*. Unlike the offline case we have discussed in Sec. III, 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.

B. Maluca*'s Performance*

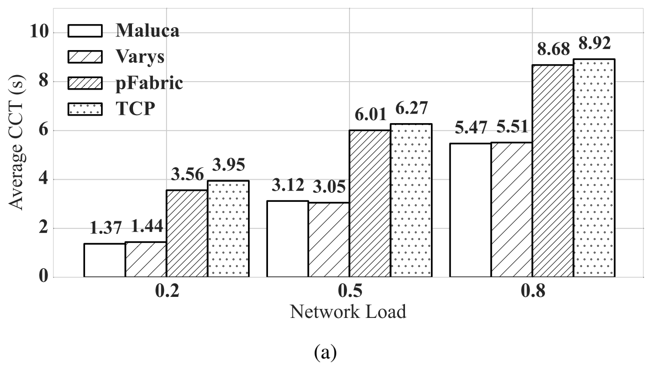
From the results in Fig. 3, we can see that *Maluca* significantly improves the average link utilization: compared to Varys, pFabric and TCP, *Maluca* improves the average link utilization by about 19.04%, 16.82% and 22.95% ($\lambda = 0.2$). It is worth noticing that Varys shows no advantage over per-flow schemes in terms of link utilization since it only guarantees work conservation without considering better utilization. As for the coflow performance (Fig. 4), due to embracing coflow semantics, *Maluca* 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. 3(a) and Fig. 4(a), the link utilizations and the coflow completion times of the four schemes increase as the network becomes busier. *Maluca* 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 *Maluca* significantly outperform per-flow mechanisms. Furthermore, *Maluca* further reduces coflow completion times when network load is high due to its ability in fully utilizing link bandwidth.

To avoid starvation, *Maluca* 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. 3(b) and 4(b). As the network load and other parameters remain the same, the performance benchmark schemes is not influenced. In contrast, the link utilization under *Maluca* 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, *Maluca* 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.



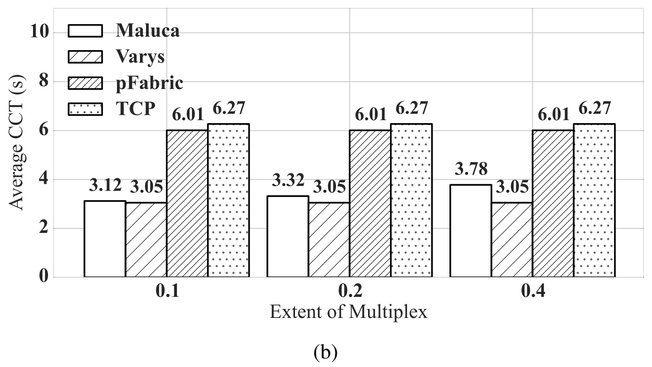


Fig. 4. The coflow completion times of four schemes under varying network loads ((a) $\alpha = 0.1$.) and varying extents of multiplexing ((b) $\lambda = 0.5$).

VI. Related Work

Researchers have made continuously efforts to maximize network throughput and improve link utilization. Recent flow-level scheduling algorithms (*e.g.*, [3]) struggle to achieve work conservation, which is insufficient for the optimal link efficiency. B4 [1] leverages 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 it effectively improves network utilization, but similar to other flow-level schemes, it still ignores 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 coflow completion time. Chowdhury *et al.* in [5] propose the smallest bottleneck first heuristic as the scheduling strategy. Aalo [9] further simplifies the coflow scheduling without the need to acquire coflow information in advance. Qiu *et al.* in [13] 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 [14], achieves high link utilization and considers the correlated demands of applications at the same time. However, it sacrifices utilization for strategy proofness.

Apart from scheduling-only mechanisms, Alizadeh *et al.* propose Conga [15], an in-network load balancing mechanism, to detect global congestion and distribute network loads more evenly. The network utilization can then be effectively improved. Zhao *et al.* in [7] try to reduce the coflow completion time by combine coflow scheduling and dynamic routing via Rapier. Leveraging OpenFlow-enabled switches, flows are routed to a centrally computed path. Nevertheless, both Conga and Rapier require upgrade of existing switches. In contrast, *Maluca* can be built upon existing network devices without too much effort.

VII. Conclusion

In this paper, we have proposed and studied the problem of maximizing link utilization with coflow-awareness. We have designed and implemented a hierarchical online scheduling mechanism, *Maluca*, to conduct two-level 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. *Maluca* 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 *Maluca* is 2-approximate when access links are homogeneous. We demonstrate that *Maluca* is able to maximize link utilization without loss of coflow performance through extensive realistic simulations.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.