Application-Aware Bandwidth Scheduling for Data Center Networks

Andrew Lester[1] Yongning Tang[2] Tibor Gyires[2]
[1]*Cloud Networking Group. Cisco Systems, Inc. - San Jose, CA, USA*
[2]*School of Information Technology. Illinois State University. Normal, IL. USA*
aeleste@cisco.com, ytang@ilstu.edu, tbgyires@ilstu.edu

***Abstract*—Recent study showed that many network applications require multiple different network flows to complete their tasks. Provisioning bandwidth to network applications other than individual flows in data center networks is becoming increasingly important to achieve user satisfaction on their received network services. Modern data center networks commonly adopt multi-rooted tree topologies. Equal-Cost Multi-Path (ECMP) forwarding is often used to achieve high link utilization and improve network throughput. Meanwhile, max-min fairness is widely used to allocate network bandwidth fairly among individual network flows. Today's data centers usually host diverse applications, which have various priorities (e.g., mission critical applications) and service level agreements (e.g., high throughput). It is unclear how to adopt ECMP forwarding and max-min fairness in the presence of such requirements. In this paper, we first propose a flow-based scheduling mechanism (called *FlowSch*) to provide a prioritized Max-Min fair multiple path forwarding to improve link utilization and improve application performance. Then, we demonstrate and discuss that *FlowSch* may not perform effectively when network applications commonly use multiple network flows to accomplish their tasks. Accordingly, we design an application-aware scheduling mechanism (called *AppSch*) to tackle this challenge. *AppSch* can optimally allocate available bandwidth to satisfy application requirements. Our performance evaluation results show that *FlowSch* can improve flow throughput 10-12% on average and increase overall link utilization especially when the total demanded bandwidth is close or even exceeds the bisectional bandwidth of a data center network. However, when most applications rely on multiple network flows, *AppSch* can improve link utilization more effectively and reduce the application completion time 36-58%.**

Keywords- application-aware; SDN; max-min fair; scheduling.

I. Introduction

Elastic cloud computing is becoming pervasive for many emerging applications, such as big data online analysis, virtual computing infrastructure, and various web applications. Various cloud applications commonly share the same network infrastructure [2] [4] [29] in a data center, and compete for the shared resource (e.g., bandwidth). Many of these emerging cloud applications are complex combinations of multiple services, and require predictable performance, high availability, and high intra-data center bandwidth. For example, Facebook "experiences 1000 times more traffic inside its data centers than it sends to and receives from outside users", and the internal traffic has increased much faster than Internet-facing bandwidth [40]. Meanwhile, many data center networks are oversubscribed, as high as $40 : 1$ in some Facebook data centers [41], causing the intra-data center traffic to contend for core bandwidth. Hence, providing bandwidth guarantees to specific applications is highly desirable, in order to preserve their response-time predictability when they compete for bandwidth with other applications.

The challenge of achieving high resource utilization makes cloud service providers under constant pressure to guarantee quality of service and increase customer satisfaction.

A Data Center (DC) refers to any large, dedicated cluster of computers that is owned and operated by a single authority, built and employed for a diverse set of purposes. Large universities and private enterprises are increasingly consolidating their Information Technology (IT) services within on-site data centers containing a few hundred to a few thousand servers. On the other hand, large online service providers, such as Google, Microsoft, and Amazon, are rapidly building geographically diverse cloud data centers, often containing more than 10,000 servers, to offer a variety of cloud-based services such as web servers, storage, search, on-line gaming. These service providers also employ some of their data centers to run large-scale data-intensive tasks, such as indexing Web pages or analyzing large data-sets, often using variations of the MapReduce paradigm.

Many data center applications (e.g., scientific computing, web search, MapReduce) require substantial bandwidth. With the growth of bandwidth demands for running various user applications, data centers also continuously scale the capacity of the network fabric for new all-to-all communication patterns, which presents a particular challenge for traditional data forwarding (switching and routing) mechanisms. For example, MapReduce based applications, as a currently adopted default computing paradigm for big data, need to perform significant data shuffling to transport the output of its map phase before proceeding with its reduce phase. Recent study shows the principle bottleneck in large-scale clusters is often inter-node communication bandwidth. Traffic pattern study [27] showed that only a subset (25% or less) of the core links often experience high utilization.

The disruptive Software-Defined Networking (SDN) technology shifts today's networks that controlled by a set of vendor specific network primitives to a new network paradigm empowered by new programmatic abstraction. OpenFlow provides a protocol such that the logical centralized controller can exploit forwarding tables on SDN switches for programmatic multi-layer forwarding flexibility. One of the fundamental transformations that flow based forwarding presents is the inclusion of multi-layer header information to make forwarding match and action logic programmatically. Programmatic policy is vital to manage the enormous combinations of user requirements. For example, an SDN controller can flexibly define a network flow using a tuple as (incoming port, MAC Src, MAC Dst, Eth Type, VLAN ID, IP Src, IP Dst, Port Src, Port Dst, Action), or schedule specific flows onto desired network paths. With the new flexibility and capability on network traffic manipulation empowered by SDN, various new network architecture and control mechanisms have been proposed for data center networks to optimize their resource allocation.

Modern data center networks commonly adopt multi-rooted tree topologies [2] [4] [29]. ECMP is often used to achieve high link utilization and improve network throughput. Meanwhile, max-min fairness is widely used to allocate network bandwidth fairly among multiple applications. Many current data center schedulers, including Hadoops Fair Scheduler [37] and Capacity Scheduler [35], Seawall [34], and DRF [38], provide max-min fairness. The attractiveness of max-min fairness stems from its generality. However, today's data centers usually host diverse applications, which have various priorities (e.g., mission critical applications) and service level agreements (e.g., high throughput). It is unclear how to adopt ECMP forwarding and max-min fairness in the presence of such requirements.

In this paper, we first propose a Flow-based Scheduling mechanism (called *FlowSch*) to provide a prioritized Max-Min fair multiple path forwarding to improve link utilization and flow-based throughput. *FlowSch* can optimally allocate current available bandwidth to satisfy user demands specified by per flow. When predefined user requirements are available, *FlowSch* can prioritize current demands and allocate available bandwidth accordingly. Then, we demonstrate and discuss that *FlowSch* may not perform effectively when network applications commonly use multiple network flows to accomplish their tasks. Accordingly, we design an Application-Aware Scheduling mechanism (called *AppSch*) to tackle this challenge. Our evaluation shows that *AppSch* can optimally allocate available bandwidth to satisfy application requirements.

The rest of the paper is organized as the following. Section II discusses the related research work. Section III describes *FlowSch*. Section IV formalizes the application-aware scheduling problem and presents our solution *AppSch*. Section V presents our simulation design and results, respectively. Finally, Section VI concludes the paper with future directions.

II. Related Work

Current large data center networks connect multiple Ethernet LANs using IP routers and run scalable routing algorithms over a number of IP routers. These layer 3 routing algorithms allow for shortest path and ECMP routing, which provide much more usable bandwidth than Ethernets spanning tree. However, the

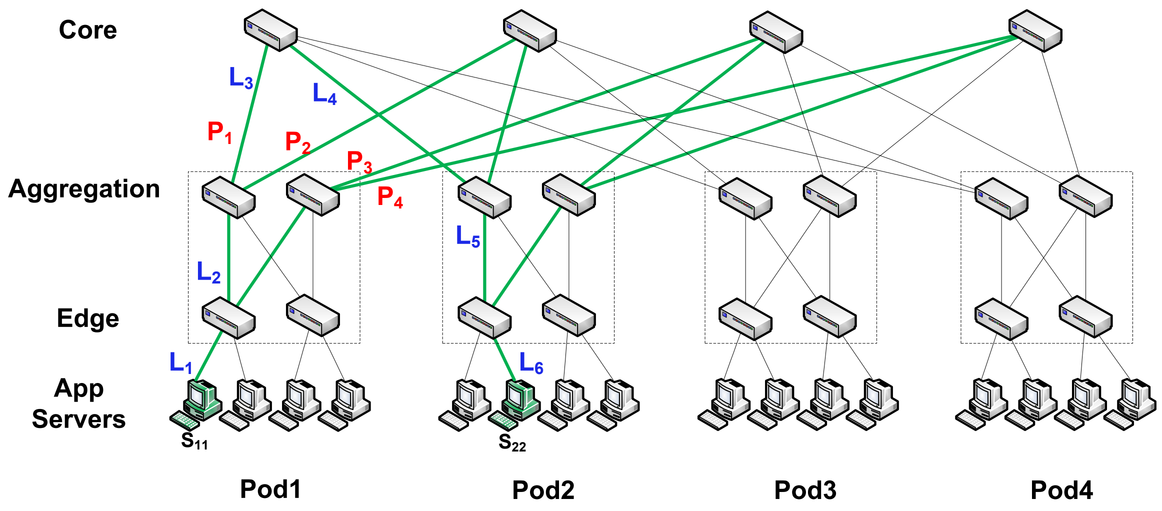


Figure 1. Fat tree topology.

mixed layer 2 and layer 3 solutions require significant manual configuration.

The trend in recent works to address these problems is to introduce special hardware and topologies. For example, PortLand [4] is implementable on Fat Tree topologies and requires ECMP hardware that is not available on every Ethernet switch. TRILL [5] introduces a new packet header format and thus requires new hardware and/or firmware features.

There have been many recent proposals for scale-out multi-path data center topologies, such as Clos networks [6] [8], direct networks like HyperX [9], Flattened Butterfly [11], DragonFly [12], etc., and even randomly connected topologies have been proposed in Jellyfish [16].

Many current proposals use ECMP-based techniques, which are inadequate to utilize all paths, or to dynamically load balance traffic. Routing proposals for these networks are limited to shortest path routing (or K-shortest path routing with Jellyfish) and end up under utilizing the network, more so in the presence of failures. While DAL routing [9] allows deroutes, it is limited to HyperX topologies. In contrast, Dahu [29] proposes a topology-independent, deployable solution for non-minimal routing that eliminates routing loops, routes around failures, and achieves high network utilization.

Hedera [17] and MicroTE [22] propose a centralized controller to schedule long lived flows on globally optimal paths. However, they operate on longer time scales and scaling them to large networks with many flows is challenging. Techniques like Hedera, which select a path for a flow based on current network conditions, suffer from a common problem: when network conditions change over time the selected path may no longer be the optimal one. While DevoFlow [24] improves the scalability through switch hardware changes, it does not support non-minimal routing or dynamic hashing. Dahu can co-exist with such techniques to better handle congestion at finer time scales.

MPTCP [19] proposes a host based approach for multi-path load balancing by splitting a flow into multiple sub flows and modulating how much data is sent over different subflows based on congestion. However, as a transport protocol, it does not have control over the network paths taken by subflows. Dahu [29] exposes the path diversity to MPTCP and enables MPTCP to efficiently utilize the non-shortest paths in a direct connect network. There have also been proposals that employ variants of switch-local per-packet traffic splitting [30].

Traffic engineering has been well studied in the context of wide area networks. TeXCP [31] and REPLEX [32] split flows on different paths based on load. However, their long control loops make them inapplicable in the data center context that requires faster response times to deal with short flows and dynamic traffic changes. PDQ [10] and pFabric [36] can support a scheduling policy like shortest flow first

(SFF), which minimizes flow completion times by assigning resources based on flow sizes. FLARE [11] exploits the inherent burstiness in TCP flows to schedule "flowlets" (bursts of packets) on different paths to reduce extensive packet reordering.

In our previous work [1], a flow-based bandwidth scheduling approach has been introduced. Several recent work [3] [7] [14] contributed to task-Aware Schedulers and network Abstractions. Orchestra [13] and CoFlow [7] argued for bringing task awareness in data centers. Orchestra focuses on how task awareness could provide benefits for MapReduce style workloads, and focuses on improvement in the average task completion time for batched workload. Baraat [3] makes the scheduling decisions in a decentralized fashion based on a revised FIFO mechanism. Baraat can also improve the tail completion time, for dynamic scenarios and multi-stage workloads. CoFlow [7] focuses on a new abstraction that can capture rich task semantics, which is orthogonal to Baraats focus on scheduling policy and the underlying mechanism. However, beyond the abstraction, CoFlow does not propose any new scheduling policy or mechanism to achieve task-awareness.

III. Flow-based Prioritized Max-Min Fair Bandwidth Scheduling

While ECMP is often used to achieve high link utilization, max-min fairness is widely used to allocate network bandwidth fairly among multiple applications. However, today's data centers usually host diverse applications, which have various priorities (e.g., mission critical applications) and service level agreements (e.g., high throughput). It is unclear how to adopt ECMP forwarding and max-min fairness in the presence of such requirements. We propose Prioritized Max-Min Fair Multiple Path forwarding (*FlowSch*) to tackle this challenge. In the following, we first formalize the problem, and then present how *FlowSch* works.

A. Problem Formalization

Consider a data center network with K-ary fat-tree topology as shown in Fig.1, composed of a set of core switches S_c, a set of aggregation switches S_a, a set of edge switches S_e, and a set of hosts H. Each switch has k-port. There are k pods. Each pod contains $k/2$ aggregation switches and $k/2$ edge switches. In each pod, each k-port edge switch is directly connected to $k/2$ hosts and $k/2$ aggregation switches. The i^{th} port of each core switch $s_i \in S_c (i \in [1, (k/2)^2])$ is connected to pod i [4]. We assume all links (e.g., L_1 in Fig.1) have the same bandwidth for both uplink (e.g., L_1^u) and downlink (e.g., L_1^d) connections.

Recent study [27] showed that less than 25% of the core links have been highly utilized while packet losses and congestions may still often occur. In this paper, we only focus on inter-pod network traffic that requires bandwidth from core links. We denote all links between aggregation and core layers as a set L_{ac}, all links between edge and aggregation layers as a set L_{ea}, and all links between application server and edge layers as a set L_{se}. Generally, in a network with K-ary fat-tree topology , there are k paths between any two hosts from different pods.

Input: A list of tasks $\{T_i\}$; current link utilization $U(L_j)$
Output: Path assignment PA with PA_i for each task T_i

```
1: Sort $\{T_i\}$ based on their priority levels $K_i$
2: Start from the highest priority $W = m$ /*$m$ is the highest priority level*/
3: **for all** $T_i! = \emptyset$ $(PL(T_i) = W)$ **do**
4:    /*The function $PL()$ returns the priority level of a given task*/
5:    Find all paths for each task $T_i$
6:    Assign a unit bandwidth (UB) to the least utilized path for each task /*we choose UB = 100Kbps*/
7:    $PA_i \leftarrow \{T_i, \{P_i\}\}$
8:    $PA \leftarrow PA \cup \{PA_i\}$
9:    **if** A path $P$ is saturated and $P \in APL(T_i)$ **then**
10:      $APL(T_i) \leftarrow APL(T_i) - P$
11:   **end if**
12:   **if** $APL(T_i) == \emptyset$ **then**
13:      Remove $T_i$
14:   **end if**
15:   **if** $(\{T_i\} == \emptyset)$ and $(W > 1)$ **then**
16:      $W = m - 1$
17:   **end if**
18: **end for**
19: return $PA$
```

Figure 2. Multi-Level progressive filling algorithm

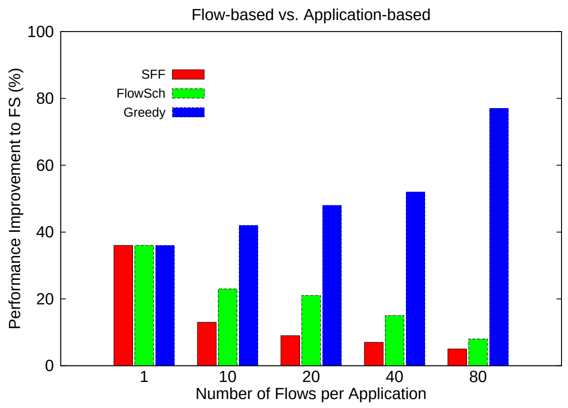


Figure 3. Completion time comparison

A network task T_i is specified by a source and destination hosts (e.g., S_{11} and S_{22}) and the expected traffic volume. We also consider each task with different priority level w_i. Here, $w_i \in [1, m]$ with the lowest and highest priority levels as 1 and m, respectively. A network scheduler modular (also simply referred to as scheduler) on an SDN controller needs to decide how to allocate available bandwidth to maximally satisfy the application requirements. We define a valid Network Path Assignment PA_i for a given task T_i is a set of paths and their corresponding allocated bandwidths connecting the source to the destination (e.g., a subset of $\{P_1, P_2, P_3, P_4\}$), in which each path consists of a list of directional links (e.g., $P_1 = \{L_1^u, L_2^u, L_3^u, L_4^d, L_5^d, L_6^d\}$) connecting the source to the destination hosts. Here, $L_1^u, L_6^d \in L_{se}$; $L_2^u, L_5^d \in L_{ea}$; $L_3^u, L_4^d \in L_{ac}$.

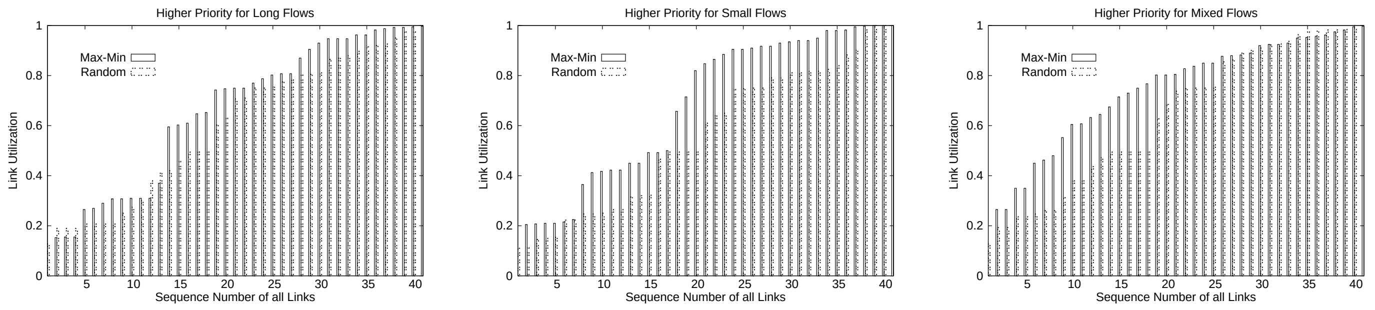
There is a variety of applications on a data center network, which have different service requirements regarding throughput, packet loss, and delay. For our analysis, we characterize the applications' requirements through their priority levels, which can be the output of some utility function. Priorities can offer a basis for providing application and business oriented service to users with diverse requirements. We consider a model where the weight associated with the different priority classes is user-definable and static. Users can freely define the priority of their traffic, but are charged accordingly by the network. We aim to study the bandwidth-sharing properties of this priority scheme. Given a set of network tasks $T = \{T_i\}$ $(i \geq 1)$ and their corresponding priority levels $K = \{K_i\}$, we consider a Network Path Assignment problem is to find a set of path assignment $PA = \{PA_i\}$ to satisfy the condition of Prioritized Max-Min Fairness.

Definition 1. Prioritized Max-Min Fairness A feasible path assignment PA^x is "prioritized max-min fair" if and only if an increase of any path bandwidth within the domain of feasible bandwidth allocations must be at the cost of a decrease of some already less allocated bandwidth from the tasks with the same or higher priority level. Formally, for any other feasible bandwidth allocation scheme PA^y, if $BW(PA^y_{T_i}) > BW(PA^x_{T_i})$, then it decreases the allocated bandwidth of some other path with the same or higher priority level. Here, $BW(PA^y_{T_i})$ is the total allocated bandwidth for the task T_i in the bandwidth allocation scheme PA^y.

Definition 2. Saturated Path A path P_i is saturated if at least one bottleneck link L_j exists on the path P_i. A link is bottlenecked if the total assigned bandwidth on this link from the given tasks is more than or equal to the maximum bandwidth of the link. Formally, a bottleneck link is the one that $\sum_i BW_{T_i}(L_j) \geq BW_{max}(L_j)$.

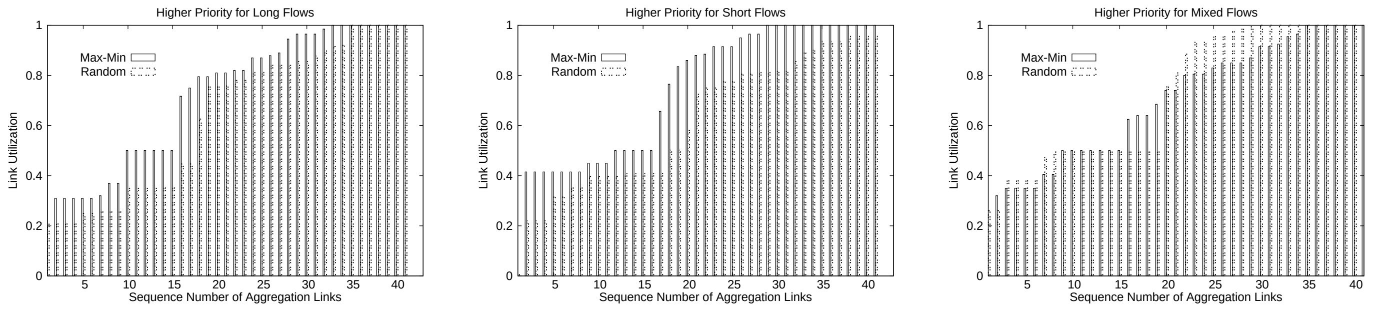
B. The Algorithm of Multi-Level Progressive Filling

The network tasks can be dynamically and continuously generated, and submitted to the scheduler. In *FlowSch*, the scheduler can periodically query all network switches to collect current link utilizations. Once a new task list received, the scheduler will use a practical approach called "progressive filling" [33] provisioning available bandwidth that results in a prioritized max-min fair allocation following the priority order from the highest to the lowest priority level. The idea is shown in Fig. 2: The scheduler starts with all provisioned bandwidth equal to 0 and increases all bandwidths together at the same pace for the tasks with the same priority level until one or several saturated paths are found. The bandwidth for the corresponding tasks that use these paths are not increased any more and the scheduler continue increasing the bandwidth for other tasks on the same priority level. All the tasks that are stopped have a saturated path. The algorithm continues until it is not possible to increase the bandwidth for the tasks at certain priority level. Then, the algorithm moves to the next priority level and repeats the same bandwidth provisioning operations until all tasks are assigned



(a) (b) (c)

Figure 4. Path utilization with higher priority for (a) long flows (b)short flows (c)mixed flows



(a) (b) (c)

Figure 5. Aggregation link utilization with higher priority for (a) long flows (b)short flows (c)mixed flows

to some paths. The algorithm terminates because the total paths and tasks are finite. When the algorithm terminates all tasks have been served at some time and thus have a saturated path. By Definition 1 the allocation is max-min fair for the tasks at the same priority level.

$$\min \sum_{k=1}^{|A|} T_k \tag{1}$$

such that:

$$\sum_{j=1}^{|P|} x_{ij}^k = 1 \tag{2}$$

$$x_{ij}^k \in \{0,1\} \tag{3}$$

$$B_j \in \{0,1\} \tag{4}$$

IV. Application-based Scheduling

Recent study [38] showed that 70% of applications involve 30-100 flows, 2% involve more than 150 flows. In a multi-flow based application, the application flows may traverse different parts of the network and not all of them may be active at the same time. Only after all these related flows finish, the corresponding application finishes and the user gets a response. The distributed nature and scale of data center applications results in rich and complex work flows. Typically, these applications run on many servers that, in order to respond to a user request, process data and communicate across the internal network. Traditionally, allocation of network bandwidth has targeted per-flow fairness. Because latency is the primary goal for many data center applications, recent proposals [21] [36] indicate that per-flow fairness scheduling that optimizes flow-level metrics (e.g., minimizing flow completion time) may not necessarily improve user perceivable application performance. Typical data center application applications can have many flows, potentially of different sizes. Flow-based scheduling presents some inefficiency when applied to applications relying on multiple flows.

In the following, we first present the result of a simple simulation that demonstrates the different effect on application performance (i.e., completion time) improvement with three different scheduling algorithms, namely Shortest Flow First

(*SFF*), *FlowSch*, and an off-line greedy scheduling algorithm (*Greedy*). Then, we tackle the inefficiency of flow-based scheduling by introducing a new application-aware scheduling approach called *AppSch*.

A. Flow-based vs. Application-based Scheduling

FlowSch is designed as an application agnostic scheduler that targets per-flow fairness among applications with the same priority, which may not improve the performance of multi-flow based applications.

We validate this through a simple simulation that compares performance improvement in terms of application completion times with three different approaches, namely Shortest Flow First (*SFF*), *FlowSch*, and an off-line greedy scheduling algorithm (*Greedy*). *SFF* schedules the shorter flows of every task first, leaving longer flows to the end. This can hurt application performance by delaying completion of tasks. *FlowSch* considers max-min fair sharing among all flows that allocates resources with a lower bound. *Greedy* is an off-line greedy algorithm searching for the “best” assignments for all flows, which also shows the room for improvement.

In this simulation, we use a simple single-stage partition-aggregate work flow scenario [7] with 60 applications comprising flows uniformly chosen from the range $[5, 40]$ KB. Fig. 5 shows SFFs improvement over fair-sharing as a function of the number of flows in a application. If an application has just a single flow, SFF reduces the application completion time by almost 40%. However, as we increase the number of flows per application, the benefits reduce. The same observation also occurred with FlowSch, which however, outperforms SFF due to its max-min sharing. Comparing to the “best” scheduling offered by the off-line greedy scheduling algorithm, application agnostic flow-based scheduling does not perform well in terms of the improvement on application completion time, comparing to the performance of an off-line application-aware scheduler referred to as *Greedy* in Fig. 5.

B. Application Requirements Abstraction

Although many data-intensive applications are network-bound [13] [17], network scheduling remains agnostic to application specific network requirements. In recent work, Coflow [7] argues for tasks (or Coflows) as a first-order abstraction for the network data plane to compensate the mismatch that often affects application-level performance, even when network-oriented metrics like flow completion time (FCT) or fairness improve. The recently proposed coflow abstraction [7] represents such collections of parallel flows to convey application-specific network requirements, for example, minimizing completion time or meeting a deadline to the network and enables application-aware network scheduling.

Allowing applications to expose their semantics to the network could significantly help the network optimize its resource allocation for application-level metrics. For example, allocating network bandwidth to applications in a FIFO fashion, such that they are scheduled over the network one at a time, can improve the average application completion time as compared to per-flow fair sharing (e.g., TCP).

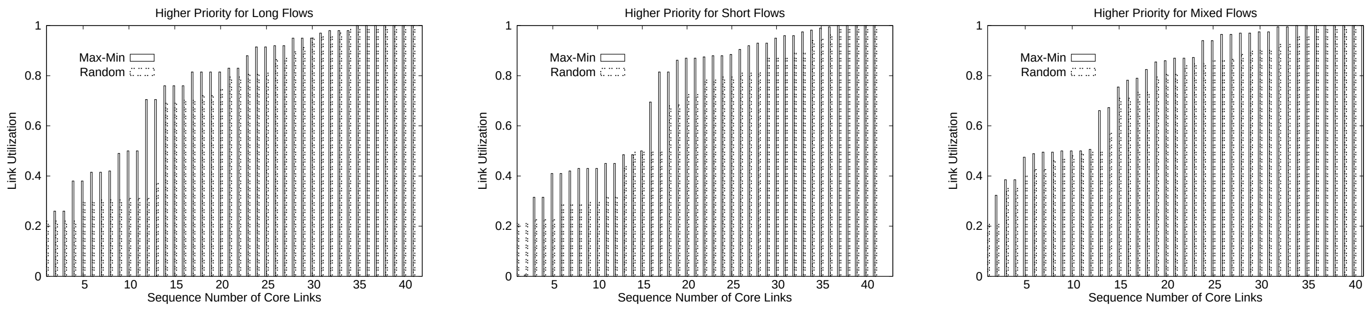
In this paper, we characterize two features of application tasks in todays data centers: 1) the task size, and 2) the number of flows per task. Both information are critical when considering application-aware scheduling for the network; the first influences the scheduling policy, while the latter governs when application-aware scheduling outperforms flow-based scheduling.

In the following, we formalize the application-aware scheduling as a variant bin packing problem, and presents a heuristic algorithm to tackle this NP-hard problem.

C. Bin Packing with Varying Capacities

We assume that the amount of data each flow in an application needs to transfer is known before it starts [13] [23] [36]. Analysis of production application traces [14] shows wide variations in application flow characteristics in terms of total size, the number of parallel flows, and the size of individual flows. But commonly these applications can be modeled as an ordered flow request list.

Let A be a list of applications A_i to be scheduled. Each application A_i has a list of flow volumes $V_i = \{v_{i1}, \cdots, v_{ik}\}$. Let $P = \{P_1, \cdots, P_m\}$ be the set of available network paths and let B_i be the current available bandwidth for path P_i. Without any loss of generality, we assume that the bandwidths associated with the network paths are integers.



(a) (b) (c)

Figure 6. Core link utilization with higher priority for (a) long flows (b)short flows (c)mixed flows

We define the path-selection variables $S = (s_1, \cdots, s_m)$, where $s_j = 1$ if path p_j is selected and $s_j = 0$, otherwise; and the flow-to-path assignment variables x_{ij}^k, where $x_{ij}^k = 1$ if flow f_{ik} from application A_k is scheduled on path P_j and $x_{ij}^k = 0$, otherwise. We want to schedule all application related flows to optimally utilize all available bandwidth, so as to minimize the total application completion time (T). For application A_k, the corresponding application completion time $T_k = \sum_{j=1}^{m} v_{ij}/(x_{ij}^k * B_j)$.

The objective function (1) minimizes the total amount of application completion time. Constraint (2) ensure that each flow request is assigned exactly to one network path; and constraints (3) and (4) enforce the integrality requirements for all decision variables.

The scheduling policy determines the order in which applications are scheduled across the network paths. Determining an ordering that minimizes application completion time can be easily reduced to a bin packing problem with varying bin sizes, which is NP-hard. Some previous similar work like flow-shop scheduling [14] [25], is considered as one of the hardest NP-hard problems, with exact solutions not known for even small instances of the problem [15]. Thus, we need to consider heuristic scheduling policies. The heuristic policy should help reduce both the average as well as tail application completion time. Guided by flow-based policies that schedule flows one at a time [17], we consider serving applications one at a time. This can help finish applications faster by reducing the amount of contention in the network. Consequently, we define application packing as the set of policies where an entire application is scheduled before moving to the next.

Specifically, for such a bin packing NP-hard problem [18], we design a heuristic algorithm called AppSch that adapts the well-known Best First Decreasing loading heuristic [15] and extends a number of fundamental concepts [15] [18] in the bin covering and knapsack methodology. AppSch first sorts all application flow requests according to the non-increasing order of their data sizes, and then sequentially assigns them into the path with the maximum available bandwidth. For each flow request, AppSch first attempts to assign it into the “best” already-selected path to increase the path utilization. If the flow request cannot be assigned to an already-selected path, a new path is selected and the flow request is assigned to it. One challenge in this problem different from classic bin packing problem where all bins are homogeneous, is how to choose a new path when required. Inspired by the item-selection rule for knapsack problems, we select paths according to the non-increasing order of the ratios of their data sizes and available path bandwidths, and in the non-decreasing order of their data sizes when the data sizes are equal.

V. Evaluation

In this section, we present our evaluation metrics, methodology and evaluation results.

A. Data Center Network Traffic Pattern

Several recent studies [26] [27] [28] have been conducted in various data center networks to understand network traffic patterns. The studied data center networks include university campus, private enterprise data centers, and cloud data centers running Web services, customer-facing applications, and intensive Map-Reduce jobs. The studies have

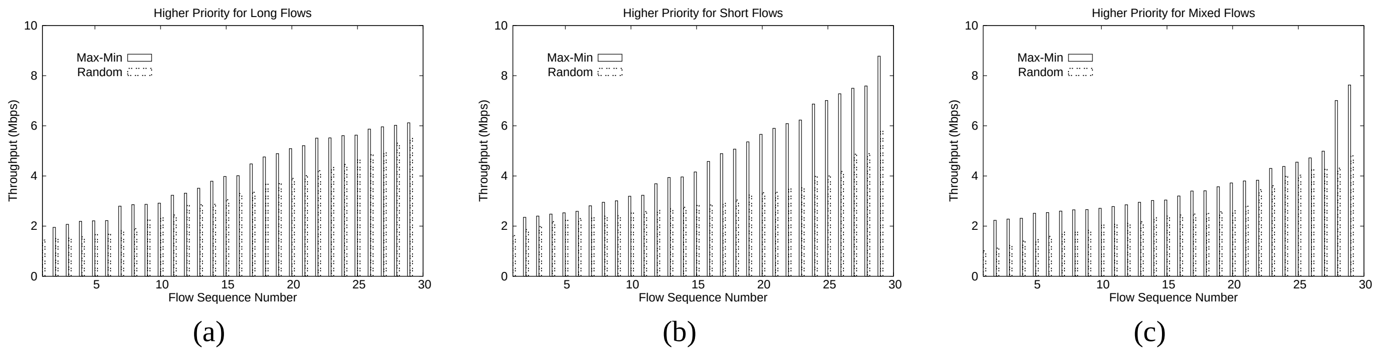


Figure 7. Throughput with higher priority for (a) long flows (b)short flows (c)mixed flows

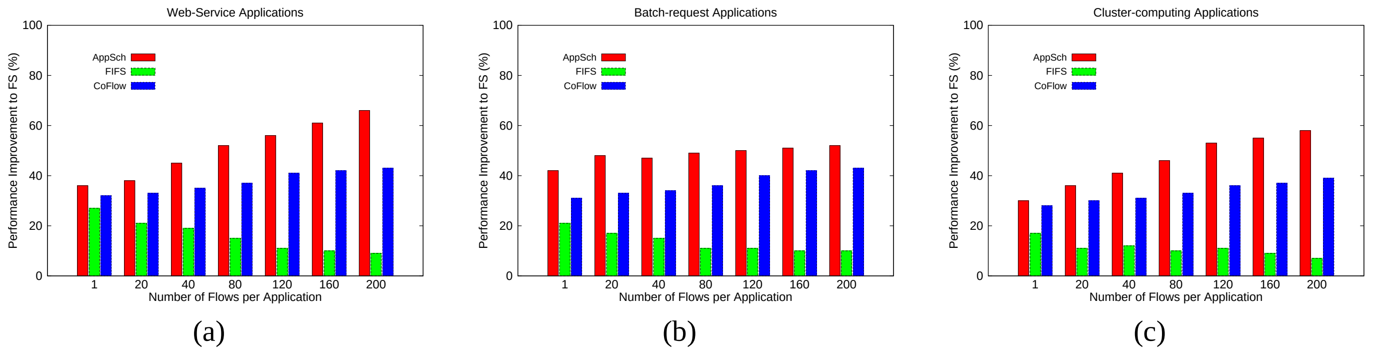


Figure 8. Application completion time for (a) Web Services (b) Batch Requests (c) Cluster Computing

shown some interesting facts: (1) The majority of the traffic in data center networks is TCP flows. (2) Most of the server generated traffic in the cloud data centers stays within a rack, while the opposite is true for campus data centers. (3) At the edge and aggregation layers, link utilizations are fairly low and show little variation. In contrast, link utilizations at the core network are high with significant variations over the course of a day. (4) In some data centers, a small but significant fraction of core links appear to be persistently congested, but there is enough spare capacity in the core to alleviate congestion. (5) Losses on the links that are lightly utilized on the average can be attributed to the bursty nature of the underlying applications run within the data centers.

B. Methodology and Metrics

In our experiments, we simulate a data center with a fat-tree topology. We implemented *FlowSch* based on RipL [39], a Python library that simplifies the creation of data center code, such as OpenFlow network controllers, simulations, or Mininet topologies. We compared *FlowSch* scheduler with a commonly used randomization based scheduling method.

In our evaluation, we use three different priority policies for a mixture of traffic patterns: (1) high priority for long TCP flows with the total data size between $1MB$ and $100MB$; (2) high priority for short TCP flows with the total data size between $10KB$ and $1MB$; (3) high priority for random selected flows including both short and long ones referred to as mixed TCP flows.

We focus on two performance metrics: (1) Link Utilization that demonstrates how effectively the scheduler utilizes the network bandwidth. Intuitively, when there are high bandwidth demands from user applications, the overall link and path utilizations should be kept in high. (2) Network throughput that shows how efficiently the network serves different applications.

For evaluating the performance of application-aware scheduler $AppSch$, we setup different application scenarios to mimics (1) web-service: a typical web-service scenario with one pod dedicated to the front-end nodes, while the other pods are used as caching back-end. For the experiment, we consider an online scenario where each user independently receives requests based on a Poisson arrival process. Each request (or application) corresponds to a multi-get that involves fetching data from randomly chosen back-end servers; (2) batched requests: to evaluate the impact of varying the number of concurrent applications in the system. For this experiment, one pod acts as a client while the other pods in the network act as storage servers. For the request, the client retrieves $100-800$ KB chunks from each of the servers. The request finishes when data is received from all servers; and (3) cluster computing: our workload is based on a Hive/MapReduce trace collected from a tier-1 ISP IDS system. We consider jobs with non-zero shuffle and divide them into bins based on the fraction of their durations spent in shuffle.

C. Link Utilization

We created 16 test scenarios to evaluate *FlowSch* with different inter-pod traffic patterns. We ran 5 tests for each scenarios. In all test scenarios, the test traffic traversed all edge, aggregation, and core links. The results of multiple test runs from the same test scenario present similar results. In the following, we only report the result of one test run for each test scenario that created traffic between two pods in both directions. Under the same three different priority policies, Fig.4(a)$\sim$(c) shows the overall path utilization; Fig.5(a)$\sim$(c) shows the aggregation link utilizations; and Fig.6(a)$\sim$(c) shows the core link utilizations. Comparing to the randomization based scheduler, our algorithm 2 achieves high utilization on path level, aggregation and core link levels by: (1) dynamically observing all link utilization status, and (2) progressively filling the jobs of the same priority with the available bandwidth with the max-min fairness. The average gain on utilization is approximately improved from 59% to 66%. Note that with the increase of link utilization, idle bandwidth can be effectively utilized by demanding network applications, which can correspondingly improve their performance by reducing their network latencies.

D. Network Throughput

Once the overall utilization can be increased, we expect that the overall application throughput should also be improved. The experiment results presented some interesting results as shown in Fig.7(a)$\sim$(c). When we emulate more realistic application scenarios, where short and long TCP flows are randomly mixed together, our *FlowSch* scheduler obviously outperforms the performance of the random scheduler with about 10-12% improvement. In the scenario of the different policies favoring either short or long flows, our scheduler adopts max-min fairness, and thus, the average throughput has been improved from $2.52Mbps$ in the random scheduler and to $3.46Mbps$ in the max-min scheduler.

E. Application Completion Time

The minimum completion time of an application A_i can be attained as long as all flows in the application finish at time T_i. We choose three applications (1) Web service, (2) Batch request, and (3) Cluster computing application that represent typical application communication patterns in today's data center networks to validate $AppSch$ and compare with $FIFS$ and $CoFlow$ [7]. From the experiment results as shown in Fig. 8, AppSch performs steadily with evident application performance (completion time) improvement (36%-58%) for all different types of applications.

VI. Conclusion

The role of the data center network is becoming ever more crucial today, which is evolving into the integrated platform for next-generation data centers. Because it is pervasive and scalable, the data center network is developing into a foundation across which information, application services and all data center resources, including servers, storage are shared, provisioned, and accessed. Modern data center networks commonly adopt multi-rooted tree topologies. ECMP is often used to achieve high link utilization and improve network throughput. Meanwhile, max-min fairness is widely used to allocate network bandwidth fairly among multiple applications. However, today's data centers usually host diverse applications, which have various priorities (e.g., mission critical applications) and service level agreements (e.g., high throughput). It is unclear how to adopt ECMP forwarding and max-min fairness in the presence of such requirements. We propose Prioritized Max-Min Fair Multiple Path forwarding (*FlowSch*) to tackle this challenge. *FlowSch* can prioritize current demands and allocate available bandwidth accordingly.

Our performance evaluation results show that *FlowSch* can improve application throughput 10-12% on average and increase overall link utilization especially when the total demanded bandwidth close or even exceed the bisectional bandwidth of a data center network.

References

[1] A. Lester, Y. Tang, T. Gyires. “Prioritized Adaptive Max-Min Fair Residual Bandwidth Allocation for Software-Defined Data Center Networks,” In the Thirteenth International Conference on Networks (ICN), 2014.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, A Scalable, “Commodity Data Center Network Architecture,” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2008

[3] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, “Decentralized Task-Aware Scheduling for Data Center Networks,” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2014

[4] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, “PortLand: A scalable fault-tolerant layer 2 data center network fabric,” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2009

[5] R. Perlman, “Rbridges: Transparent routing,” In IEEE Conference on Computer Communications (INFOCOM), 2004.

[6] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “VL2: A Scalable And Flexible Data Center Network,” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2009.

[7] M. Chowdhury and I. Stoica, “Coflow: A networking abstraction for cluster applications,” In ACM Hot Topics in Networks (HotNets) workshops, 2012.

[8] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, “F10: A Fault-Tolerant Engineered Network,” In USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2013

[9] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, “HyperX: Topology, Routing, and Packaging of Efficient Large-Scale Networks,” In ACM Conference on High Performance Computing Networking, Storage and Analysis, 2009.

[10] C. Hong, M. Caesar, and P. Godfrey, “Finishing flows quickly with preemptive scheduling,” ACM SIGCOMM Computer Communication Review (CCR), 2012.

[11] J. Kim, W. J. Dally, and D. Abts, “Flattened butterfly: A Cost-efficient Topology for High-radix networks,” In ACM International Symposium on Computer Architecture (ISCA), 2007.

[12] J. Kim, W. J. Dally, S. Scott, and D. Abts, “Technology-Driven, Highly-Scalable Dragonfly Topology,” In ACM International Symposium on Computer Architecture (ISCA), 2008

[13] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, “Managing data transfers in computer clusters with Orchestra,” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2011.

[14] M. Chowdhury, Y Zhong, and I. Stoica, “Efficient Coflow Scheduling with Varys,” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2014.

[15] W T Rhee and M Talagrand, “Optimal bin covering with items of random size,” SIAM Journal on Computing. 1989.

[16] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, “Jellyfish: Networking Data Centers Randomly,” In USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2012.

[17] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks,” In USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2010.

[18] A. S. Fukunaga and R. E. Korf, “Bin Completion Algorithms for Multicontainer Packing, Knapsack, and Covering Problems,” Journal of Artificial Intelligence Research 28 (2007) pp. 393 $\sim$ 429

[19] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center TCP (DCTCP),” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2010.

[20] T. Benson, A. Akella, and D. A. Maltz, “Network Traffic Characteristics of Data Centers in the Wild,” In ACM Internet Measurement Conference (IMC), 2010.

[21] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, “Better never than late: Meeting deadlines in datacenter networks,” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2011.

[22] T. Benson, A. Anand, A. Akella, and M. Zhang, “MicroTE: Fine Grained Traffic Engineering for Data Centers,” In ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2011.

[23] C.-Y. Hong, M. Caesar, and P. B. Godfrey, “Finishing flows quickly with preemptive scheduling,” In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2012.

[24] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula,

P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-Performance Networks," In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2011.

[25] OpenFlow Switch Specification (Version 1.1), www.openflow.org/documents/openflow-spec-v1.1.0.pdf, (retrieved: Nov. 2014)

[26] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements and Analysis," In ACM Internet Measurement Conference (IMC), 2009.

[27] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," In ACM Internet Measurement Conference (IMC), 2010.

[28] G. Wang, D. G. Andersen, M. Kaminsky, M. Kozuch, T. S. E. Ng, K. Papagiannaki, M. Glick, and L. Mummert, "Your data center is a router: The case for reconfigurable optical circuit switched paths," In ACM Hot Topics in Networks (HotNets) workshops, 2009.

[29] S. Radhakrishnan, M. Tewari, R. Kapoor, G. Porter, and A. Vahdat, "Dahu: Commodity Switches for Direct Connect Data Center Networks," In Proceedings of the 9th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS13), October 2013

[30] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2012.

[31] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2005.

[32] S. Fischer, N. Kammenhuber, and A. Feldmann, "REPLEX: Dynamic Traffic Engineering Based on Wardrop Routing Policies," In ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2006.

[33] A. Ghodsi, M. Zaharia, S. Shenker and I. Stoica, "Choosy: Max-Min Fair Sharing for Datacenter Jobs with Constraints," In European Conference on Computer Systems (EuroSys), 2013.

[34] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, I. Stoica, and S. Shenker, "Dominant resource fairness: Fair allocation of multiple resource types," In USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011

[35] Hadoop Capacity Scheduler, hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html, (retrieved: Nov. 2014)

[36] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," In ACM conference of the Special Interest Group on Data Communication (SIGCOMM), 2013.

[37] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," In European Conference on Computer Systems (EuroSys), 2010.

[38] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," In USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011.

[39] M. Casado, D. Erickson, I. A. Ganichev, R. Griffith, B. Heller, N. Mckeown, D. Moon, T. Koponen, S. Shenker, and K. Zarifis, "Ripcord: A modular platform for data center networking," UC, Berkeley, Technical Report UCB/EECS-2010-93

[40] "Facebook Future-Proofs Data Center With Revamped Network," http://tinyurl.com/6v5pswv, (retrieved: Nov. 2014)

[41] N. Farrington and A. Andreyev, "Facebook's Data Center Network Architecture," IEEE Optical Interconnects, 2013.