Barrier-Aware Max-Min Fair Bandwidth Sharing and Path Selection in Datacenter Networks

Li Chen[1] Baochun Li[1] Bo Li[2]
[1]University of Toronto
[2]The Hong Kong University of Science and Technology

***Abstract*—In production datacenters operated by Web service providers such as Google, multiple data parallel applications, such as MapReduce, are employed to facilitate data processing at a large scale, with a strong demand for intra-datacenter bandwidth in their communication stages. A noteworthy phenomenon in these applications is the presence of *barriers*, which implies that a job will not finish until the last task completes. Existing flow-level sharing in datacenter networks is not designed and optimized to meet such application-level needs. In this paper, we promote the awareness of application barriers in the design of both bandwidth allocation and path selection strategies. In particular, we propose the notion of *application-level* fairness when bandwidth is allocated, with favorable properties of performance-centric max-min fairness and Pareto efficiency. Further, we show that both application-level performance and resource utilization can be further improved by considering path selection as well. With our implementation in the Mininet emulation testbed and large-scale simulations, we demonstrate that our new barrier-aware strategy for fair bandwidth sharing and path selection significantly outperforms barrier-agnostic strategy when application performance is concerned.**

I. Introduction

In current production datacenters operated by Web service providers such as Google and Facebook, a wide variety of data-intensive applications are hosted for large-scale data analytics, ranging from PageRank [1] to machine learning [2]. To efficiently process large volumes of data, these applications typically embrace data parallel frameworks to proceed in multiple computation stages, each with a group of computation tasks running in parallel. Between successive computation stages, the intermediate data is transferred over the datacenter network, called the *communication stage*. With MapReduce [3] as an example, during the communication stage, the intermediate data produced in the *map* stage is *shuffled* over the network to be processed by *reduce* computation tasks.

A common characteristic shared by most of the existing data parallel frameworks is the existence of *barriers* [4], which implies that the application would not finish or progress until the completion of all its constituent flows. To be more specific, a MapReduce job would not finish until its last reducer finishes, exhibiting the phenomenon of a *barrier* at the end of the job. Naturally, barriers exist in frameworks (*e.g.*, Pig [5]) that use MapReduce as its building block. Pregel [6], a framework that proceeds in a series of *supersteps*, contains *barriers* at the end of each *superstep*.

In the communication stage of a data parallel application, there is a strong demand for intra-datacenter bandwidth by the constituent flows, in order to complete the data transfer as soon as possible. With the presence of *barriers*, the application performance, represented by the reciprocal of the amount of time required to complete the data transfer, is determined by the slowest flow. Therefore, to achieve better utilization, bandwidth allocation should be performed at the application level, so that all the constituent flows of an application would finish at the same time. Blindly allocating more bandwidth to a faster flow does not make any improvement to the application performance.

As multiple data parallel applications share the same datacenter network, a commonly accepted manner is to share the bandwidth *fairly*. Specially customized for data parallel applications with barriers, *performance-centric fairness* [7] regulates that fairness should be maintained with respect to the performance achieved by each application. The guiding principle of such notion of fairness is that the application performance should be proportional to their weights. However, bandwidth allocation with such fairness constraint fails to achieve efficiency, since some available bandwidth is left unused to keep the strict performance proportionality.

In this paper, we propose the notion of *application-level fairness* as the guiding principle for the bandwidth allocation among data parallel applications with barriers. With the satisfaction of such fairness, among all the applications, the performance achieved is weighted max-min fair [8], an implication of which is that the worst performance is optimized. Also, Pareto efficiency [9] can be satisfied, since no application can improve its performance without degrading the performance of others.

There is further space to improve the worst performance, as well as bandwidth utilization, if we are allowed the flexibility to select paths in multi-rooted tree topologies [10]–[12], which is the norm in current datacenters. The simple intuition is that the performance of an application can be improved if its slowest flow shifts from a heavily-loaded path to a lightly-loaded one. Existing effort in path selection is either barrier-aware but orthogonal to fair sharing of link bandwidth [13], or coupled with fair bandwidth sharing but agnostic to the

The co-authors would like to acknowledge the gracious research support from the NSERC Discovery Research Program and the SAVI NSERC Strategic Networks Grant at the University of Toronto, the grants from RGC under the contracts 615613 and 16211715, and the grant from NSFC and Guangdong joint project under the contract U1301253.

presence of barriers [14]. To fill this gap, we wish to optimize the path selection for all the concurrent flows, yet with the fair bandwidth allocation jointly considered, which remains an open challenge.

In this paper, we begin with the problem formulation of bandwidth allocation with the constraint of application-level fairness, given the paths traversed by all the flows. With a theoretical study of this problem, we design an algorithm and explore its properties. Then, with path selection also considered as decision variables, the problem is extended to a joint optimization that has a larger solution space.

The upshot in this paper revolves around a joint optimization of bandwidth allocation and path selection, with the objective of maximizing the worst application performance, while achieving application-level fairness. It turns out that this problem is equivalent to a Mixed-Integer Linear Program, which is NP-hard [15]. Therefore, we develop a practical greedy-based algorithm to approximate the maximum worst performance, while satisfying the proposed fairness.

With the rising popularity of software-defined networking (SDN) [16], the network can be conveniently tailored to meet application demands through a well-defined programming interface such as OpenFlow [17]. We thus implement our algorithm in a centralized controller in the Mininet emulation testbed, with the global view of network states for our optimization. Compared with barrier-agnostic strategy for fair sharing and path selection [14], our barrier-aware strategy achieves better bandwidth utilization and application performance. Moreover, extensive large-scale simulations demonstrate the effectiveness of our strategy towards fairness satisfaction and performance maximization.

The remainder of this paper is organized as follows. In Sec. II, we introduce the motivation and objective of our barrier-aware strategy, with respect to both bandwidth allocation and path selection. In Sec. III, we formulate the problem of bandwidth allocation, with known flow paths. We then propose a new algorithm and study its properties theoretically. Further, we formulate the joint optimization problem when path selection is to be decided, and design a greedy-based algorithm in Sec. IV. Our experimental results are presented in Sec. V, and we conclude the paper in Sec. VII.

II. Motivation and Design Objectives

Our design of barrier-aware strategy for joint bandwidth allocation and path selection is motivated by both the awareness of application barriers and the requirement of application-level fairness.

A. Barriers Awareness in Bandwidth Allocation

With the presence of a *barrier* at the end of a computation stage, the performance of an application depends on the time when the last computation task completes. Without loss of generality, the computation times of parallel tasks can be considered the same, due to mature techniques of load balancing and resource allocation. In contrast, the completion times of flows in the communication stage are quite variable since the network resource is globally shared, with bottleneck links contended by flows from multiple applications.

To achieve efficiency in bandwidth utilization, all the flows of a communication stage should finish at the same time. To understand the reason, let us consider a flow of an application that will complete earlier than other flows. If we reduce its bandwidth to let it complete at the same time with other flows, the performance of this application will remain the same. However, the saved bandwidth can be better utilized by the slowest flow of another application to improve its performance. Therefore, bandwidth should be allocated at the application level, with awareness of application barriers, in order to avoid wasting bandwidth.

In particular, within an application, the amount of bandwidth allocated to each flow should be of the same proportionality to the volume of data to be transmitted, so that all of these flows can finish at the same time. Across concurrent applications, bandwidth should be allocated so that the performance of these applications achieves weighted max-min fairness. We refer to such a notion of fairness as *application-level fairness*, which is quite different from existing efforts. First, without barrier awareness, traditional allocation strategies decide the amount of bandwidth for each flow in an isolated way, with the belief that the more the bandwidth, the better the performance. In contrast, we have shown that a larger amount of bandwidth allocated to a single flow does not necessarily improve the application performance. Second, max-min fairness is traditionally defined for the rates achieved by concurrent flows. On the contrary, in our barrier-aware bandwidth allocation, max-min fairness is defined for the performance achieved by concurrent applications (or groups of flows).

We next provide an example to better illustrate our basic idea of barrier-aware bandwidth allocation with the satisfaction of the proposed fairness.

Coupling within an application. We first consider the allocation of bandwidth at a bottleneck link shared by two flows, represented as *A1-A2* and *B1-B2*, from different applications. Application *A* has another flow *A1-A3* which is expected to finish within $10s$; application *B* has another flow *B1-B3* which is expected to finish within $8s$. The bandwidth capacity of the shared link is 15 MB/s, and the amounts of intermediate data transmitted by *A1-A2* and *B1-B2* are 50 MB and 100 MB, respectively.

Traditional flow-level bandwidth allocation, such as TCP, would allocate the same amount (7.5 MB/s) of bandwidth to the contending flows, in order to achieve flow level fairness. With such an allocation, *A1-A2* will finish within $\frac{20}{3}s$, while the flow completion time of *B1-B2* is $10s$. Since barriers exist in both applications, the performance is determined by the slowest flow completion time, which we refer to as the *transfer time*. Hence, the transfer time of *A* is $10s$, with *A1-A3* as the slowest flow, while the transfer time of *B* is also $10s$, with *B1-B2* as the slowest flow. In this sense, a part of the bandwidth allocated to *A1-A2* is wasted, without improving the performance of *A*.

In contrast, if we use barrier-aware allocation, the bandwidth

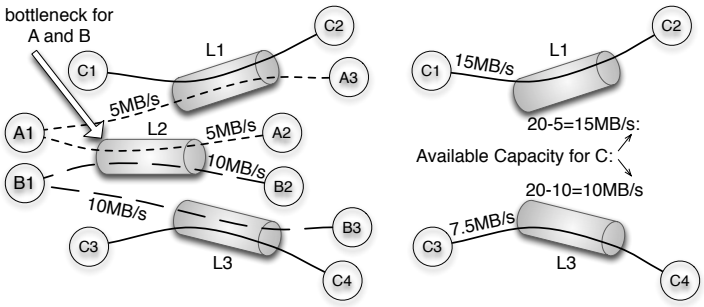


Fig. 1: An illustration of bandwidth allocation to achieve application-level fairness among applications *A*, *B* and *C*.

allocated to flows within an application should be coupled. In this example, *B1-B2* requires 12.5 MB/s to finish within $8s$, while *A1-A2* can utilize the residual bandwidth to complete within $10s$. This way, bandwidth is more efficiently used to improve the performance at the application level. We can see that, compared with flow-level sharing that is barrier-agnostic, the performance of *B* is improved to $8s$, without degrading performance of *A*.

Max-min fairness across applications. Now we consider bandwidth allocation across multiple applications, still with the awareness of barriers. As shown in Fig. 1, link *L1*, *L2*, and *L3*, with a capacity of 20 MB/s, 15 MB/s and 20 MB/s, respectively, are shared by the flows in *A*, *B*, and *C*. Each of the two flows in application *A* transmits 50 MB of intermediate data; each of the flows in *B* transmits 100 MB. *A1-A3* shares link *L1* with *C1-C2*, which needs to transmit 80 MB data, while *B1-B3* shares link *L3* with *C3-C4*, and the amount of data to be sent is 40 MB.

In this allocation, *L2* is first bottlenecked, with 5 MB/s allocated to *A1-A2* and 10 MB/s allocated to *B1-B2*. Due to the coupling relationship among flows of the same application, the amounts of bandwidth allocated to *A1-A3* and *B1-B3* are also determined as 5 MB/s and 10 MB/s. Thus, both *A* and *B* achieve the same application performance.

After the allocations of *A* and *B* have been fixed, the available capacities for *L1* and *L3* are reduced as 15 MB/s and 10 MB/s, respectively. Then *C1-C2* and *C3-C4* can be allocated the residual bandwidth still in a coupled way, so that they both finish at the same time, as shown on the right side of Fig. 1. (Note that despite some idle bandwidth in *L3*, allocating it to *C3-C4* does not improve the application performance.) In this way, max-min fairness is achieved among *A*, *B* and *C*, with respect to their application-level performance.

B. Flexibility in Path Selection

Given our barrier-aware fair bandwidth allocation, the worst application performance is determined by the degree of contention at the bottleneck link. Intuitively, if the link load is well balanced, both the worst application performance and bandwidth utilization can be optimized. As multiple equal-cost paths exist in today's datacenters with multi-rooted tree topologies [10]–[12], through flexible path selection, we can balance the link load to achieve the optimum.

A datacenter network with a fat-tree topology [10] is shown in Fig. 2. For the flow that is sent from server *S1* and received by *R1*, the selected path shown in the figure can be represented as *FCBDH*. If the flow *S2-R2* passes along the path *GCBEI*, it will share the bottleneck link *C-B* with the flow *S1-R1*. Suppose all the links have capacities of 100 MB/s, and these flows are identical[1]. According to our allocation, both of these flows will send at 50 MB/s. If flow *S2-R2* switches its path to *GCAEI*, the link load will be balanced, with link *C-B* no longer being the bottleneck. In this case, both flows can send at 100 MB/s, so that the application performance is improved and bandwidth utilization is increased.

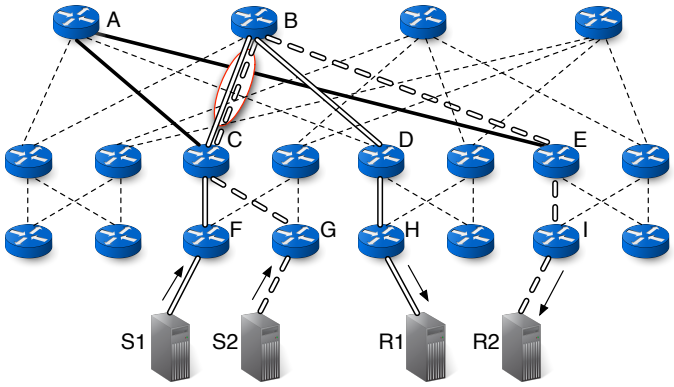


Fig. 2: An illustration of path selection resulting in different application performance and bandwidth utilization.

III. Barrier-Aware Bandwidth Allocation with Max-Min Fairness

Before we jointly consider barrier-aware fair bandwidth allocation and path selection, we first investigate the problem of bandwidth allocation with application-level fairness, given the paths taken by all the flows.

A. Problem Formulation

In a private datacenter, the network is shared by K data-parallel applications with active transfers. Each application $k \in \mathcal{K} = \{1, 2, ..., K\}$ is represented by $(\mathcal{F}_k, w_k)$, where $\mathcal{F}_k = \{1, 2, ...n_k\}$ denotes the set of its flows whose total number is n_k, and w_k represents the weight of the application. For each flow $i \in \mathcal{F}_k$, the amount of data to be sent is d_{ki}.

In a datacenter with a typical multi-rooted tree topology, flow $i \in \mathcal{F}_k$ can exploit a set of equal-cost paths, denoted as $\mathcal{P}_{ki}$. In our setting, a flow cannot be split across multiple paths, to avoid packet reordering. Let the binary variable $\mathcal{I}_{ki}^j$ denote whether flow i chooses path $j \in \mathcal{P}_{ki}$, *i.e.*,

$$\mathcal{I}_{ki}^j = \begin{cases} 1, & \text{when flow } i \in \mathcal{F}_k \text{ chooses path } j \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Similarly, the binary variable $h_{ki}^{jl} \in \{0, 1\}$ denotes whether link $l \in \mathcal{L}$ is on path $j \in \mathcal{P}_{ki}$.

[1]This example only highlights our problem from a particular perspective; a more thorough investigation of the problem is presented in Sec. IV.

Let r_{ki} denote the amount of bandwidth allocated to flow i of application k. The total amount of bandwidth to be allocated at link l is $\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{F}_k}\sum_{j\in\mathcal{P}_{ki}} r_{ki}\mathcal{I}^j_{ki}h^{jl}_{ki}$, which represents the sum of the allocated bandwidth over all the flows that pass through this link. More specifically, the bandwidth allocated to flow $i \in \mathcal{F}_k$ is added, if the flow follows path $j \in \mathcal{P}_{ki}$, *i.e.*, $\mathcal{I}^j_{ki} = 1$, and link l is on this path, *i.e.*, $h^{jl}_{ki} = 1$. Let C_l denote the bandwidth capacity of link l, then the link capacity constraint is represented as:

$$\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{F}_k}\sum_{j\in\mathcal{P}_{ki}} r_{ki}\mathcal{I}^j_{ki}h^{jl}_{ki} \le C_l, \forall l \in \mathcal{L} \tag{2}$$

For application k, the performance metric α_k is defined as:

$$\alpha_k = 1/t_k, \tag{3}$$

where t_k represents the transfer time that is determined by the slowest flow, *i.e.*, $t_k = \min_{i\in\mathcal{F}_k} d_{ki}/r_{ki}$, as implied by the existence of a barrier. Obviously, in order to avoid wasting bandwidth, all the constituent flows of a transfer should finish at the same time. Hence, the bandwidth allocated to flows within an application should be in proportion to the volume of data to be sent by them, *i.e.*, $r_{ki} \propto d_{ki}, \forall i \in \mathcal{F}_k$. As a result, t_k can be represented as follows:

$$t_k = d_{ki}/r_{ki}, \forall i \in \mathcal{F}_k \tag{4}$$

To allocate bandwidth across applications, we first illustrate rigorously that *performance-centric fairness* [7] does not achieve bandwidth efficiency. Such fairness regulates the weight proportionality with respect to application performance, represented as:

$$\alpha_k \propto w_k, \forall k \in \mathcal{K}, \tag{5}$$

Substituting Eq. (3) and (4) into Eq. (5) yields:

$$r_{ki}/d_{ki} \propto w_k \Rightarrow r_{ki} \propto w_k d_{ki}, \forall i \in \mathcal{F}_k, k \in \mathcal{K},$$

This equation has the following equivalent form:

$$r_{ki} = \lambda w_k d_{ki}, \forall i \in \mathcal{F}_k, k \in \mathcal{K}, \tag{6}$$

where λ is referred to as the *allocation factor*, which is constrained by link capacities. The problem of bandwidth allocation with *performance-centric fairness* can thus be formulated as:

$$\begin{aligned} \max_{\lambda} \quad & \lambda \\ \text{s.t.} \quad & \text{Eq. (2)}, \quad \text{Eq. (6)} \end{aligned} \tag{7}$$

which can be equivalently represented as:

$$\begin{aligned} \max_{\lambda} \quad & \lambda \\ \text{s.t.} \quad & \lambda \le C_l/(\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{F}_k}\sum_{j\in\mathcal{P}_{ki}} w_k d_{ki}\mathcal{I}^j_{ki}h^{jl}_{ki}), \forall l \in \mathcal{L} \end{aligned}$$

Obviously, the optimal solution can be derived as:

$$\lambda^* = \min_{l\in\mathcal{L}} C_l/(\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{F}_k}\sum_{j\in\mathcal{P}_{ki}} w_k d_{ki}\mathcal{I}^j_{ki}h^{jl}_{ki}) \tag{8}$$

With λ obtained, the amount of bandwidth to be allocated to each flow is computed according to Eq. (6). Analyzing this allocation, we obtain the following proposition:

***Proposition** 1:* Bandwidth allocation with *performance-centric fairness*, as described in Problem (7), is not able to achieve the best utilization of bandwidth.

The basic idea of the proof is to show that there exists an application $\bar{k}$, such that for all the links that flows of $\bar{k}$ pass through, there is idle bandwidth, which could have been utilized by application $\bar{k}$ to improve its performance. However, since bandwidth allocation across all the applications is coupled through a common allocation factor λ, according to Eq. (6), application $\bar{k}$ is not allowed to utilize more available bandwidth, and thus efficiency is not achieved.

Next, we study our *application-level fairness* in bandwidth allocation, and demonstrate its favorable properties of performance maximization and bandwidth efficiency.

To achieve such a fairness, we relax Eq. (6) as $r_{ki} = \lambda_k w_k d_{ki}, \forall i \in \mathcal{F}_k, k \in \mathcal{K}$, where λ_k is referred to as the *allocation factor* for application k. We then present the following definitions as the basis of our problem formulation.

***Definition** 1:* $\langle\boldsymbol{\lambda}\rangle = (\langle\lambda\rangle_1, \langle\lambda\rangle_2, ..., \langle\lambda\rangle_K)$ denotes a version of vector $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_K)$ with its elements sorted in non-decreasing order. To be more rigorous, $\langle\boldsymbol{\lambda}\rangle$ satisfies $\langle\lambda\rangle_1 \le \langle\lambda\rangle_2 \le ... \le \langle\lambda\rangle_K$, where $\langle\lambda\rangle_k = \lambda_{\varphi(k)}$ and φ is a permutation on the set $\{1, 2, ..., K\}$.

***Definition** 2:* For two vectors $\boldsymbol{x} \in \mathbb{R}^K$ and $\boldsymbol{y} \in \mathbb{R}^K$, if $\exists k \in \{1, 2, ..., K\}$ such that $x_k > y_k$ and $x_n = y_n, \forall n \in \{1, ..., k-1\}$, $\boldsymbol{x}$ is *lexicographically greater* than $\boldsymbol{y}$.

***Definition** 3:* The lexicographical maximization is to maximize the non-decreasingly sorted vector $\langle\boldsymbol{\lambda}\rangle$, represented as:

$$\text{lexmax}_{\boldsymbol{\lambda}}\langle\boldsymbol{\lambda}\rangle \;\Rightarrow\; \text{lexmax}_{\boldsymbol{\lambda}}(\langle\lambda\rangle_1, \langle\lambda\rangle_2, ..., \langle\lambda\rangle_K)$$

We are now ready to formulate the problem of barrier-aware bandwidth allocation as follows:

$$\begin{aligned} \text{lexmax}_{\boldsymbol{\lambda}} \quad & (\langle\lambda\rangle_1, \langle\lambda\rangle_2, ..., \langle\lambda\rangle_K) \\ \text{s.t.} \quad & \text{Eq. (2)}, \quad \text{Eq. (6)} \end{aligned} \tag{9}$$

B. Algorithm Design

Solving the lexicographical maximization in Problem (9) is equivalent to solving the following problem (associated with round n) over a decreasing solution space ($\mathcal{K}(n)$ and $\mathcal{L}(n)$) by rounds:

$$\begin{aligned} \max_{\lambda} \quad & \lambda(n) \\ \text{s.t.} \quad & \sum_{k\in\mathcal{K}(n)}\sum_{i\in\mathcal{F}_k}\sum_{j\in\mathcal{P}_{ki}} r_{ki}\mathcal{I}^j_{ki}h^{jl}_{ki} \le C_l(n), \forall l \in \mathcal{L}(n) \\ & r_{ki} = \lambda w_k d_{ki}, \ \forall i \in \mathcal{F}_k, k \in \mathcal{K}(n) \end{aligned} \tag{10}$$

In each round n, we maximize $\lambda(n)$ across all the applications in $\mathcal{K}(n)$ and allocate bandwidth to the applications that are bottlenecked at this round. These applications are then removed from $\mathcal{K}(n)$, so that Problem (10) will be solved over a smaller set of applications in the next round. This procedure iterates until all the applications have been allocated.

Based on this idea, we present our barrier-aware bandwidth allocation strategy in Algorithm 1. The input consists of flow traffic volumes, application weights, flow paths and link capacities, which can be obtained in practice (to be discussed in

Algorithm 1: Barrier-Aware Bandwidth Allocation.

Input:
Traffic volume $d_{k,i}$, Application weight w_k,
Link capacity C_l, Flow paths $\mathcal{I}^j_{k,i}$, Path-link map h^{jl}_{ki},
$\forall i \in \mathcal{F}_k,\ \forall k \in \mathcal{K},\ \forall l \in \mathcal{L}$;
Output:
Bandwidth allocation for all applications: r_{ki};
1: Initialize the set of applications: $\mathcal{K} = \{1, 2, ..., K\}$;
2: Initialize the set of links $\mathcal{L}$ and the available bandwidth $c_l = C_l,\ \forall l \in \mathcal{L}$;
3: **while** $\mathcal{K} \neq \varnothing$ **do**
4: $a_l = 0, \forall l \in \mathcal{L}$;
5: **for all** link $l \in \mathcal{L}$ **do**
6: **for all** application $k \in \mathcal{K}$ **do**
7: **for all** flow $i \in \mathcal{F}_k$ **do**
8: **if** flow i traverses link l: $\mathcal{I}^j_{ki} h^{jl}_{ki} == 1$ **then**
9: $a_l = a_l + w_k d_{k,i}$;
10: $b_l = c_l / a_l$;
11: $\lambda^* = \min_{l \in \mathcal{L}} b_l$, which is the optimal λ^* for Problem (10) across $\mathcal{K}$;
12: Obtain *Bottleneck Links* $l \in \mathcal{L}^B$ that satisfies: $b_l = \lambda^*$;
13: Obtain *Bottleneck Applications* $k \in \mathcal{K}^B$, containing the flows that traverse the bottleneck links;
14: **for all** application $k \in \mathcal{K}^B$ **do**
15: **for all** flow $i \in \mathcal{F}_k$ **do**
16: Allocate bandwidth as: $r_{ki} = \lambda^* w_k d_{ki}$;
17: **for all** link $l \in \mathcal{L}$ **do**
18: **if** flow i traverses link l: $\mathcal{I}^j_{ki} h^{jl}_{ki} == 1$ **then**
19: $c_l = c_l - r_{ki}$;
20: $\mathcal{K} = \mathcal{K} - \mathcal{K}^B$, $\mathcal{L} = \mathcal{L} - \mathcal{L}^B$;

the next section). After the initialization of the application set $\mathcal{K}$, the link set $\mathcal{L}$ and the available link bandwidth $c_l, \forall l \in \mathcal{L}$, we iteratively allocate bandwidth with the following steps (for round n), until $\mathcal{K}$ becomes empty:

1) Solve Problem (10) across applications in $\mathcal{K}(n)$ and links in $\mathcal{L}(n)$, and obtain the solution as $\lambda^*(n) = \min_{l \in \mathcal{L}(n)} b_l(n)$. (*Lines 4 – 11*)

2) Obtain the *Bottleneck Links* ($\mathcal{L}^B(n)$) that have the smallest $b_l(n)$, and the *Bottleneck Applications* ($\mathcal{K}^B(n)$) that have at least a flow passing through a bottleneck link. (*Lines 12 – 13*)

3) Allocate bandwidth to all the flows of the *Bottleneck Applications* ($\mathcal{K}^B(n)$): $r_{ki} = \lambda^*(n) w_k d_{ki},\ \forall i \in \mathcal{F}_k, k \in \mathcal{K}^B(n)$. (*Lines 14 – 19*)

4) Update available bandwidth of links in $\mathcal{L}(n)$: $C_l(n+1) = C_l(n) - \sum_{k \in \mathcal{K}^B(n)} \sum_{i \in \mathcal{F}_k} \sum_{j \in \mathcal{P}_{ki}} r_{ki} \mathcal{I}^j_{ki} h^{jl}_{ki}$. (*Lines 14 – 19*)

5) Update $\mathcal{K}$ and links $\mathcal{L}$ by removing applications and links that are bottlenecked in this round: $\mathcal{K}(n+1) = \mathcal{K}(n) - \mathcal{K}^B(n)$, $\mathcal{L}(n+1) = \mathcal{L}(n) - \mathcal{L}^B(n)$. (*Line 20*)

This algorithm has the following favorable properties:

***Property** 1:* The optimum λ^* obtained at each round is increasing.

Proof: Let $\lambda^*(n)$ represent the optimum λ^* obtained at the n-th round. Let l_β and l_γ denote the *bottleneck links* at any two consecutive rounds n and $n+1$, respectively. Let b_l represent the term $C_l / (\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{F}_k} \sum_{j \in \mathcal{P}_{ki}} w_k d_{ki} \mathcal{I}^j_{ki} h^{jl}_{ki})$. As λ^* is determined by the contention at the *bottleneck link* for each round, *i.e.*, $\lambda^* = \min_{l \in \mathcal{L}} b_l$, we have $\lambda^*(n) = b_{l_\beta}(n)$ for round n, and $\lambda^*(n+1) = b_{l_\gamma}(n+1)$ for round $n+1$.

At n-th round, l_γ is not yet bottlenecked, so we have $\lambda^*(n) < b_{l_\gamma}(n)$. Let $\mathcal{K}^B(n)$ denote the *bottleneck applications* that have their flows passing through the *bottleneck link* l_β. For ease of expression, we denote $A(n) = \sum_{k \in \mathcal{K}^B(n)} \sum_{i \in \mathcal{F}_k} \sum_{j \in \mathcal{P}_{ki}} r_{ki} \mathcal{I}^j_{ki} h^{jl_\gamma}_{ki}$, representing the total demand of all the *bottleneck applications* at link l_γ, and $B(n) = \sum_{k \in (\mathcal{K}(n) - \mathcal{K}^B(n))} \sum_{i \in \mathcal{F}_k} \sum_{j \in \mathcal{P}_{ki}} r_{ki} \mathcal{I}^j_{ki} h^{jl_\gamma}_{ki}$, representing the total demand of other applications at l_γ. Then we can represent $b_{l_\gamma}(n)$ as $C_{l_\gamma}(n)/(A(n)+B(n))$, based on which we have $\lambda^*(n) < \frac{C_{l_\gamma}(n)}{A(n)+B(n)}$ and further

$$\lambda^*(n) < (C_{l_\gamma}(n) - \lambda^*(n) A(n)) / B(n) \tag{11}$$

At $(n+1)$-th round, link l_γ is bottlenecked, thus $\lambda^*(n+1) = b_{l_\gamma}(n+1) = C_{l_\gamma}(n+1)/(A(n+1)+B(n+1))$. Since l_γ is the *bottleneck link* at this round, all the applications that have their flows passing through l_γ are *bottlenecked applications*. Thus we have $B(n+1) = 0$. At the end of the previous round, the *bottleneck applications* $\mathcal{K}^B(n)$ have been removed from $\mathcal{K}(n)$, with their flows allocated bandwidth according to $\lambda^*(n)$. Therefore, at round $n+1$, the bandwidth capacity of link l_γ has been updated as $C_{l_\gamma}(n+1) = C_{l_\gamma}(n) - \lambda^*(n) A(n)$. The applications that are not bottlenecked at the previous round are now bottlenecked, so that $A(n+1) = B(n)$. Therefore, from $\lambda^*(n+1) = \frac{C_{l_\gamma}(n+1)}{A(n+1)+B(n+1)}$ we have

$$\lambda^*(n+1) = (C_{l_\gamma}(n) - \lambda^*(n) A(n)) / B(n) \tag{12}$$

With Eq. (11) and (12), it is obvious that $\lambda^*(n) < \lambda^*(n+1)$ and we are done with the proof. ■

***Property** 2:* Barrier-aware bandwidth allocation (Problem (9)) achieves weighted max-min fairness and Pareto efficiency with respect to application performance.

Proof: At any round n, flows of *bottleneck applications* in $\mathcal{K}^B(n)$ are allocated bandwidth according to $r_{ki} = \lambda^*(n) w_k d_{ki},\ \ \forall k \in \mathcal{K}^B(n)$. For any $k_1, k_2 \in \mathcal{K}^B(n)$ that are competing for bandwidth at the *bottleneck link*, the performance metric of k_1 and k_2 are calculated as $\alpha_k = 1/t_{k_1} = \lambda^*(n) w_{k_1}$ and $\alpha_{k_2} = 1/t_{k_2} = \lambda^*(n) w_{k_2}$, respectively. Hence, we have $\alpha_{k_1} : \alpha_{k_2} = w_{k_1} : w_{k_2}$. This implies that for any competing applications within a round, their performance achieved are proportional to their weights.

The algorithm maximizes the minimum weighted performance across rounds, and the maximum possible *allocation factor* λ^* increases with rounds, as shown in *Property* 1. Across rounds, applications are gradually bottlenecked, allocated with bandwidth and ruled out from the solution space. When the algorithm terminates, weighted max-min fairness is achieved among application performance. Also, Pareto efficiency is achieved, since it is impossible to improve the performance of any one individual application without making at least one individual application worse off. ■

The time complexity of Algorithm 1 is $O(LFK^2)$, where L is the number of network links, F is the maximum number of constituent flows across all the applications, and K is the number of applications. With any two sets of input fixed, the

complexity is linearly or at most quadratically increasing with the other input, which is reasonable for resource allocation in a private datacenter.

IV. Joint Path Selection and Bandwidth Allocation

A. Problem Formulation

In this section, we move a step further to consider path selection as decisions to be made in our optimization. Specifically, $\mathcal{I}_{ki}^j$ is considered as a variable with the following constraint:

$$\mathcal{I}_{ki}^j \in \{0,1\},\ \sum_{j\in\mathcal{P}_{ki}} \mathcal{I}_{ki}^j = 1,\ \forall i \in \mathcal{F}_k, k \in \mathcal{K}, j \in \mathcal{P}_{ki} \quad (13)$$

which indicates that each flow should choose a single path from its available ones. Hence, the problem of joint barrier-aware bandwidth allocation and path selection can be formulated as:

$$\text{lexmax}_{\boldsymbol{\lambda}} \quad \langle \boldsymbol{\lambda} \rangle \qquad \text{s.t.} \quad \text{Eq. (2)},\ \text{Eq. (6)},\ \text{Eq. (13)}$$

With the flexibility afforded by allowing path selection, we now have a larger solution space than Problem (9), and it is thus possible to achieve a larger optimum $\boldsymbol{\lambda}^*$.

Similar to the previous section, we can solve this lexicographical maximization problem by iteratively solving the following single-objective joint optimization:

$$\max_{\lambda,\boldsymbol{\mathcal{I}}} \quad \lambda \qquad \text{s.t.} \quad \text{Eq. (2)},\ \text{Eq. (6)},\ \text{Eq. (13)}$$

To be particular, in the first round, we can obtain the maximum λ as $\langle\lambda\rangle_1^*$, with the optimal path selection for the problem above. If such a solution is unique, then the flow paths are selected accordingly, thus we can allocate bandwidth iteratively as in Algorithm 1 with the known paths. Otherwise, we solve this problem in the next round, over a smaller set of applications, with the flow paths selected according to the multiple optimal solutions, respectively. The solution that gives the maximum λ is the optimal path selection for our original problem (lexicographical maximization). Then, we can obtain the other $\langle\lambda\rangle_i^*$ in rounds, during which bandwidth is allocated to bottlenecked applications, as in Algorithm 1.

However, the aforementioned single-objective joint optimization is NP-hard, as we illustrate next. With λ replaced by $\nu = 1/\lambda$, it is transformed as:

$$\begin{aligned} \min_{\nu,\boldsymbol{\mathcal{I}}} \quad & \nu \\ \text{s.t.} \quad & \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{F}_k} \sum_{j\in\mathcal{P}_{ki}} w_k d_{ki} \mathcal{I}_{ki}^j h_{ki}^{jl} \le \nu C_l,\ \text{Eq. (13)} \end{aligned} \quad (14)$$

Among its variables, $\mathcal{I}_{ki}^j$ is binary while ν is continuous. The objective function and all the link capacity constraints are linear. Hence, this problem is a Mixed-Integer Linear Programming (MILP) problem, which is NP-hard [15]. Therefore, we will design a practical heuristic in the next subsection.

Algorithm 2: Barrier-Aware Strategy (Joint Bandwidth Allocation and Path Selection)

1: Initialize the total *weighted traffic volume* on each link: $E_l = 0, \forall l \in \mathcal{L}$;
2: **for all** application $k \in \mathcal{K}$ **do**
3: **for all** flow $i \in \mathcal{F}_k$ **do**
4: $tmp = 10000$; $s = 0$;
5: **for all** path $j \in \mathcal{P}_{ki}$ **do**
6: **if** $\max_{l,h_{ki}^{jl}\neq 0} (E_l + w_k d_{ki})/C_l < tmp$ **then**
7: $tmp = \max_{l,h_{ki}^{jl}\neq 0} (E_l + w_k d_{ki})/C_l$;
8: $s = j$;
9: Select path s for flow $i \in \mathcal{F}_k$: $\mathcal{I}_{ki}^s = 1$;
10: Update E_l for all the links along the selected path s: $E_l = E_l + w_k d_{ki}, \forall l$, *s.t.* $h_{ki}^{sl} \neq 0$;
11: Use Algorithm 1 to compute the barrier-aware bandwidth allocation that achieves max-min fairness.

B. Barrier-Aware Strategy

We first transform Problem (14) to the following form that is more convenient for us to understand and derive insights:

$$\begin{aligned} \min_{\boldsymbol{\mathcal{I}}} \quad & \max_{l\in\mathcal{L}} \left(\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{F}_k} \sum_{j\in\mathcal{P}_{ki}} w_k d_{ki} \mathcal{I}_{ki}^j h_{ki}^{jl}\right)/C_l \\ \text{s.t.} \quad & \text{Eq. (13)} \end{aligned}$$

The term $w_k d_{ki}$ represents the *weighted traffic volume* of flow $i \in \mathcal{F}_k$, and $\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{F}_k} \sum_{j\in\mathcal{P}_{ki}} w_k d_{ki} \mathcal{I}_{ki}^j h_{ki}^{jl}$ represents the total *weighted traffic volume* of all the passing flows at link l. If flow $i \in \mathcal{F}_k$ chooses a path $j \in \mathcal{P}_{ki}$ (*i.e.*, $\mathcal{I}_{ki}^j = 1$), then for every link l along the path (*i.e.*, l that satisfies $h_{ki}^{jl} = 1$), the total *weighted traffic volume* at this link is increased by the *weighted traffic volume* of flow i.

Interpreted in this way, the objective of our barrier-aware strategy is to select paths for all the flows, so that the resulted maximum value of the total *weighted traffic volume* at a link divided by its capacity, among all the links in $\mathcal{L}$, is minimized.

Based on this observation, we propose Algorithm 2 to jointly allocate bandwidth and choose paths for flows with the awareness of application barriers. The main idea is to greedily map each flow i to the path j that can result in a minimum value of $\max_{l\in\mathcal{L}} \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{F}_k} \sum_{j\in\mathcal{P}_{ki}} w_k d_{ki} \mathcal{I}_{ki}^j h_{ki}^{jl}/C_l$.

As illustrated in Algorithm 2, we first initialize the total *weighted traffic volume* at each link l, represented by E_l, as 0. When considering a path j for flow $i \in \mathcal{F}_k$, we calculate $\max_{l,h_{ki}^{jl}\neq 0} (E_l + w_k d_{ki})/C_l$, which represents the maximum value of $(E_l + w_k d_{ki})/C_l$ among all the links l along path j. For the flow i, the path $j \in \mathcal{P}_{ki}$ that has the smallest value of $\max_{l,h_{ki}^{jl}\neq 0} (E_l + w_k d_{ki})/C_l$ will be found through *Lines 4–8* and selected for flow i by *Line 9*. Then, the total *weighted traffic volume* at each link l along the selected path, denoted as E_l, will be updated by adding the *weighted traffic volume* of flow i, as shown in *Line 10*. This process iterates across all the flows from all the applications.

After all the flows have been mapped to their respective paths, the worst performance among all the applications is nearly optimized. With all the variables $\mathcal{I}_{ki}^j$ becoming known as input, we can further use Algorithm 1 to achieve weighted

max-min fairness with respect to application performance, and improve bandwidth utilization.

C. Implementation Aspects

With the increasing popularity of SDN and data intensive applications, it is increasingly common for a SDN controller to coordinate network transfers of these applications based on their demands, which are either conveyed from application managers [18], [19], or anticipated by the SDN controller [20].

Our idea of incorporating barrier awareness into bandwidth allocation and path selection is aligned with this line of research. Each application manager can leverage existing APIs and messages [19] to interact with the network, by sending their traffic matrices to the controller, and receiving decisions from the controller. With a global view of both application requirements and network states, the SDN controller is able to make the optimized decision on allocating bandwidth and selecting paths for all the application flows. The decision of routing will be enforced by installing rules for corresponding flows in switches, while the decision of bandwidth allocation will be conveyed to application managers for enforcement. Our assumption is that applications report real demands to the controller, and follow decisions from the controller. This is reasonable in a private datacenter without malicious competition among applications.

V. Performance Evaluation

In this section, we evaluate our barrier-aware strategy (Algorithm 2), henceforth referred to as *AppFair*, with both emulations in the Mininet 2.0 testbed [21] and extensive simulations. We demonstrate whether our strategy achieves performance max-min fairness among all the applications, and extensively investigate how the algorithm performs in maximizing the overall application performance in various settings. We also evaluate the runtime performance of our algorithm in different scales with various parameters.

A. Mininet Emulation

Experimental Setup. *AppFair* involves both path selection and bandwidth allocation, the enforcement of which requires routing flows to selected paths, and controlling flow sending rates, respectively, according to the computed results.

We implement *AppFair* in the centralized `POX` controller for a fat-tree datacenter network emulated in Mininet. Based on a global view of the network states, *AppFair* computes the paths and bandwidth allocated to all the application flows. Application flows between communicating tasks are emulated with their rates specified by the results of *AppFair*. Moreover, *AppFair* installs routing rules for these flows in the switches, according to the computed flow paths. In comparison, *Hedera* [14], the state-of-the-art barrier-agnostic strategy for fair bandwidth sharing and path selection, only determines flow paths in the centralized controller, leaving rate allocation decisions to TCP at the senders. We measure the actual bandwidth achieved by each flow, with *AppFair* and *Hedera* applied respectively, based on which the performance achieved by each application can be calculated for comparison.

Performance Comparison. We first evaluate *AppFair* over an emulated network with 4-pods fat-tree topology, which is shared by applications *A, B* and *C*. Without the loss of generality, *A* has four flows transmitting data of $300, 150, 300, 300$ MB, respectively, *B* has two flows, each sending 300 MB of data, and *C* has three flows sized 300, 300 and 100 MB, respectively. Fig. 3 presents the transfer times (each averaged over 15 times) achieved by the three applications, with *AppFair* and *Hedera* applied, respectively. Compared with *Hedera*, all the applications complete their transfers faster with *AppFair*, which demonstrates the effectiveness of *AppFair* in improving the overall application performance.

To clearly illustrate the reason for such performance improvements, we compare the bandwidth achieved by each individual flow in Fig. 4. With *AppFair*, all the flows of *A* finish at the same time, as the bandwidth allocated to each individual flow has the same proportionality $(2 : 1 : 2 : 2)$ with that of their traffic volumes. In contrast, when *Hedera* is applied, flow *A-2* transfers 150 MB data at the rate of about 0.95 MB/s, while *A-1*, with a larger size of 300 MB, obtains a smaller amount of bandwidth (0.68 MB/s), which becomes the slowest to complete, determining the application performance. In this sense, *A-2, A-3* and *A-4* only require $0.34, 0.68$ and 0.68 MB/s, respectively, to finish at the same time with *A-1* and keep the same application performance. Therefore, the excessive bandwidth beyond these amounts obtained by them is wasted. The same analysis applies to applications *B* and *C*, which demonstrates that with barrier awareness, *AppFair* can efficiently utilize bandwidth to maximize application performance, outperforming *Hedera* significantly.

B. Large-Scale Simulations

1) Tuning Parameters: Having conducted verification and insightful analysis in the Mininet testbed, we further evaluate *AppFair* by extensive simulations with various settings.

The Number of Applications. We conduct three groups of simulations in a 16-pods network, with the number of applications set as $15, 100$ and 300, respectively, while the total number of flows is 1500. We randomly set the size of each individual flow in the range of 200 MB to 400 MB. In each group, *AppFair*, *Hedera* and *BwAlloc* are applied, respectively, where *BwAlloc* represents Algorithm 1 that only allocates bandwidth according to the application-level fairness, with flows following the same paths with those in *Hedera*. The final application performance achieved by these algorithms for each test group is plotted as the empirical CDF shown in Fig. 6.

For each group, *AppFair* achieves the best application performance, corresponding to the leftmost curve in the CDF, while *Hedera* performs the worst. *BwAlloc* outperforms *Hedera*, since it has the same path selection, yet a barrier-aware bandwidth allocation to achieve better efficiency. However, it performs worse than *AppFair*, since its path selection is not jointly optimized with barrier-aware bandwidth allocation.

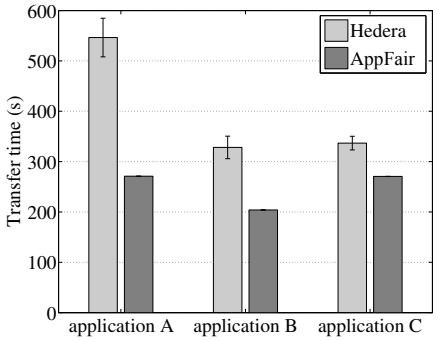


Fig. 3: Comparison between the performance (transfer times) achieved by *Hedera* and *AppFair*, for applications *A, B* and *C*.

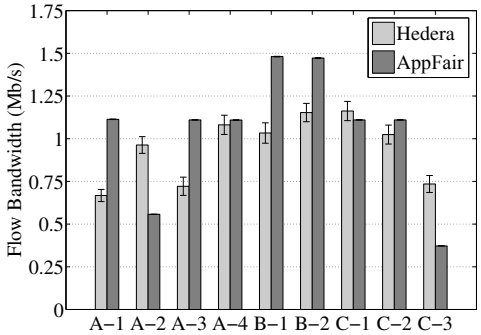
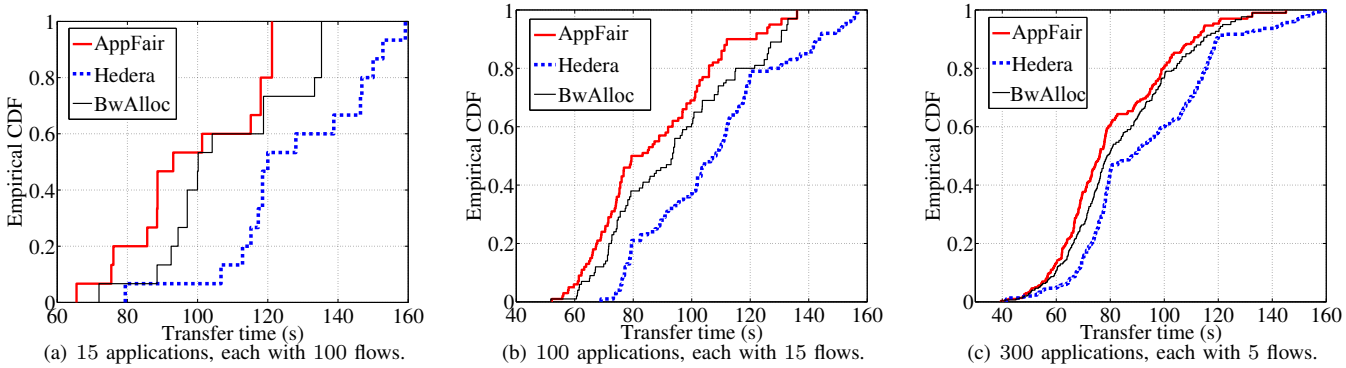


Fig. 4: Comparison between the flow bandwidth allocated by *Hedera* and *AppFair*, for 9 flows that belong to applications *A, B* and *C*.

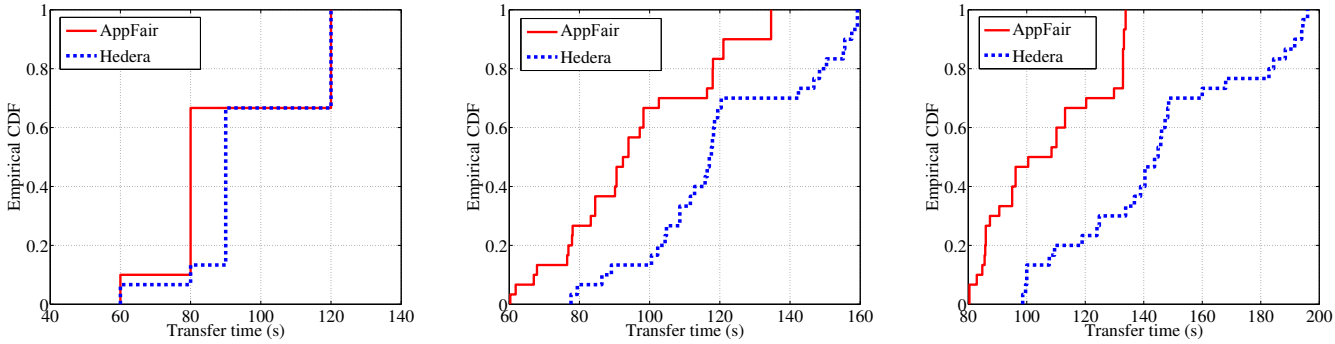
	d: (10-50)	d: (20-40)	d: (25-35)	d: 30
5 * 300	1.370	1.348	1.366	1.298
15 * 100	1.449	1.473	1.469	1.306
30 * 50	1.649	1.638	1.632	1.323
50 * 30	1.786	1.817	1.800	1.351
150 * 10	2.260	2.331	2.362	1.336

Fig. 5: Running time (seconds) of *AppFair* with different settings of traffic volumes and application numbers. 5×300 represents 5 applications, each with 300 flows.



(a) 15 applications, each with 100 flows. (b) 100 applications, each with 15 flows. (c) 300 applications, each with 5 flows.

Fig. 6: Empirical CDFs for application performance achieved by *AppFair*, *BwAlloc* and *Hedera*. Traffic volume d is randomly selected from 200 to 400.



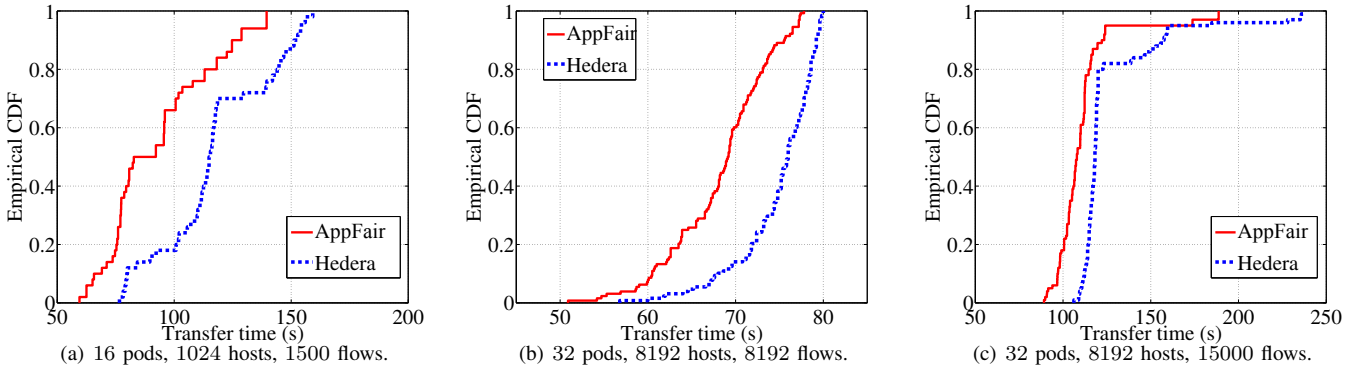
(a) $d = 300$ (b) d is randomly selected from 200 to 400. (c) d is randomly selected from 100 to 500.

Fig. 7: Empirical CDFs for application performance achieved by *AppFair* and *Hedera*. 30 applications are considered, each with 50 flows.

Comparing the three CDFs, we observe that the difference among the algorithms is decreasing, with an increasing number of applications. The reason is that, with more applications, the number of flows within each application becomes fewer (since the total number of flows is the same), which indicates weaker coupling among application flows. As a result, there is less bandwidth wastage, thus the benefit of barrier-awareness becomes limited. Intuitively, if the number of flows of each application decreases to 1, application-level fairness in *AppFair* and *BwAlloc* becomes identical to flow-level fairness in *Hedera*. This analysis implies that *AppFair* has more advantage in the case where each application has more flows.

Traffic Volume. Now we tune the amount of traffic to be sent by each flow, with the number of applications set as 30, each has 50 flows. As illustrated in Fig. 7(a), when we increase the range of traffic volumes (d), *AppFair* exhibits an increasing extent of performance improvement over *Hedera*. This observation can be briefly explained as follows: a larger variance of d may result in a larger amount of bandwidth wasted with flow-level allocation in *Hedera*, thus the performance improvement of *AppFair* with barrier awareness would be more significant.

Network Size. We further compare *AppFair* to *Hedera* with an increasing problem scale. In particular, the size of network is increased from 16 pods to 32 pods, and the total number of application flows is gradually increased from 1500



(a) 16 pods, 1024 hosts, 1500 flows. (b) 32 pods, 8192 hosts, 8192 flows. (c) 32 pods, 8192 hosts, 15000 flows.

Fig. 8: Empirical CDFs for application performance achieved by *AppFair* and *Hedera*, with different network scales.

to 15000. Without loss of generality, d is randomly selected from $[200, 400]$. The performance comparison of *AppFair* and *Hedera* is shown in Fig. 8, which demonstrates that *AppFair* scales well, and always performs better than *Hedera* in maximizing the overall application performance.

2) Running Time: Finally, we evaluate the running time of *AppFair* on a 1.86 GHz dual-core Intel Core 2 server, with different settings of the simulated datacenter network. Fig. 5 presents the running time achieved in a 16-pods fat-tree network, with the range of the traffic volume d shrinking from $[10, 50]$ to 30 horizontally, and the number of applications increasing from 5 to 150 vertically. Each time shown in the table is averaged over 15 runs. The running time is observed to increase with an increasing number of applications, because with more applications, Algorithm 1 as a component of *AppFair* would require more iterations to complete. In each row, the running time for $d = 30$ is much smaller than the other three, simply because bandwidth computation would be much easier and allocation would be finished with fewer iterations if all the flows have the same traffic volume.

	16 pods, 1500 flows	32 pods, 8192 flows	32 pods, 15000 flows
BwAlloc	0.463	10.476	16.709
AppFair	1.638	35.050	67.766
Hedera	3.625	46.975	91.335

Fig. 9: Running time comparison with different problem scales. 1500 flows belong to 30 applications, each with 50 flows, *i.e.*, $30*50$. Similarly, 8192 flows: $128 * 64$, 15000 flows: $300 * 50$.

Further, we compare the running times of *BwAlloc*, *AppFair* and *Hedera* with an increasing problem scale. Note that the difference between the running time of *AppFair* and *BwAlloc* is the time for computing flow paths, which increases with the size of the network and the total number of flows. As shown in Fig. 9, the running time of *BwAlloc* is much smaller than that of *AppFair*, which implies that path selection is more time consuming. Compared with *Hedera*, *AppFair* has a shorter running time[2] than *Hedera*, regardless of the problem scale.

[2]The running times are tens of seconds in our simulations, but in a standard datacenter server, the running times would be significantly decreased.

VI. Related Work

Fair Bandwidth Allocation. Bandwidth allocation among multiple tenants in public cloud datacenters has received a substantial amount of recent research attention [22]–[27], with the focus on ensuring fair allocation among competing tenant virtual machines (VMs) [23] or VM-pairs [24] according to their payments. However, in a private datacenter shared by data paralllel applications with barriers, bandwidth allocation with such kind of fairness does not ensure a fair share of the application performance. Agnostic of application barriers, it also fails to achieve optimal application performance and bandwidth efficiency.

In the context of a private datacenter, Kumar *et al.* proposed that bandwidth should be allocated with the awareness of communication patterns in data parallel applications [28], but they did not give a clear definition of fairness with respect to application performance in a sharing environment. Our previous work [7] proposed *performance-centric fairness* to offer a definitive guide to the problem of bandwidth allocation among multiple data parallel applications in a private datacenter. Yet, there is a tradeoff between such fairness and bandwidth efficiency.

Application Awareness. Orchestra [29] is the first work that considers the important characteristic of barriers in optimizing the data transfers within data parallel applications. It allocated bandwidth to each flow in a shuffle using weighted fair sharing, where the flow weight is proportional to the volume of data to be sent by this flow. Chowdhury *et al.* [4] proposed the concept of a *coflow* as a collection of flows between successive computation stages, which accounted for the application semantics. It was pointed out that a coflow, rather than an individual flow, should be the basic unit for network optimization. With such application awareness, existing works [13], [30]–[32] have adopted the concept of coflow in their scheduling and routing strategies. However, all these works are based on flow ordering, which are orthogonal to fair sharing considered in our work.

Multi-Path Routing. In today's datacenter networks with multi-rooted tree topologies, such as fat-tree topologies, multiple equal-cost paths exist between each pair of cross-rack

hosts. ECMP [33] exploited such path diversity by hashing. Without a global view of link states and flow statistics, ECMP may result in network congestion if two large flows are hashed to the same path. Hedera [14] and microTE [34] tried to overcome this limitation by taking advantage of a global view to dynamically make flow path decisions for load balancing. However, they both made flow-level optimization without accounting for application barriers, and as such failed to achieve application-level optimality with respect to application performance and bandwidth efficiency. RAPIER [13] and App-Sch [32] proposed coflow-aware path selection that considered application barriers, but their bandwidth allocation failed to achieve application-level fairness among sharing coflows.

VII. Concluding Remarks

In this paper, we focus on incorporating application-level characteristics when making bandwidth allocation and routing decisions in datacenter networks, so that fairness can be achieved among sharing applications, and application performance as well as bandwidth efficiency can be optimized. As the first step, we theoretically study the problem of barrier-aware bandwidth allocation, with flow paths considered as the known input, and propose an algorithm to achieve weighted max-min fairness across application performance, as well as Pareto efficiency. Then, with the flexibility of path selection, we investigate the joint optimization problem, and design an barrier-aware strategy to improve application performance and bandwidth utilization. Using Mininet emulation and extensive simulations, we compare our barrier-aware fair bandwidth allocation and path selection algorithm with barrier-agnostic strategy, which demonstrates the effectiveness of our algorithm in improving application performance and bandwidth utilization. We also evaluate our algorithm with various settings of parameters.

References

[1] A. Langville and C. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.

[2] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-Scale Parallel Collaborative Filtering for The Netflix Prize," in *Algorithmic Aspects in Information and Management*. Springer, 2008, pp. 337–348.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[4] M. Chowdhury and I. Stoica, "Coflow: A Networking Abstraction for Cluster Applications," in *Proc. ACM SIGCOMM HotNet Workshop*, 2012.

[5] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *Proc. ACM SIGMOD*, 2008.

[6] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A System for Large-Scale Graph Processing," in *Proc. ACM SIGMOD*, 2010.

[7] L. Chen, Y. Feng, B. Li, and B. Li, "Towards Performance-Centric Fairness in Datacenter Networks," in *Proc. IEEE INFOCOM*, 2014.

[8] J. Jaffe, "Bottleneck Flow Control," *IEEE Trans. Communications*, vol. 29, no. 7, pp. 954–962, 1981.

[9] R. Boadway and N. Bruce, *Welfare Economics*. Blackwell Oxford, 1984.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proc. ACM SIGCOMM*, 2008.

[11] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proc. ACM SIGCOMM*, 2009.

[12] R. Niranjan 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 *Proc. ACM SIGCOMM*, 2009.

[13] Y. Zhao, K. Chen, W. Bai, Y. Minlan, 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.

[14] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2010.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York, 1979, vol. 174.

[16] N. McKeown, "Software-Defined Networking," *INFOCOM keynote talk*, 2009.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[18] G. Wang, T. Ng, and A. Shaikh, "Programming Your Network at Run-Time for Big Data Applications," in *Proc. ACM HotSDN Workshop*, 2012.

[19] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory Networking: an API for Application Control of SDNs," in *Proc. ACM SIGCOMM*, 2013.

[20] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and C. Yu, "Transparent and Flexible Network Management for Big Data Processing in the Cloud," in *Proc. USENIX HotCloud Workshop*, 2013.

[21] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-Based Emulation," in *Proc. 8th ACM CoNEXT*, 2012.

[22] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "NetShare: Virtualizing Data Center Networks across Services," UCSD-CSE, Tech. Rep. CS2010-0957, May 2010.

[23] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the Data Center Network," in *Proc. USENIX NSDI*, 2011.

[24] 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.

[25] J. Guo, F. Liu, D. Zeng, J. Lui, and H. Jin, "A Cooperative Game Based Allocation for Sharing Data Center Networks," in *Proc. IEEE INFOCOM*, 2013.

[26] 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.

[27] L. Popa, P. Yalagandula, S. Banerjee, and J. Mogul, "ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing," in *Proc. ACM SIGCOMM*, 2013.

[28] G. Kumar, M. Chowdhury, S. Ratnasamy, and I. Stoica, "A Case for Performance-Centric Network Allocation," in *Proc. USENIX HotCloud Workshop*, 2012.

[29] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing Data Transfers in Computer Clusters with Orchestra," in *Proc. ACM SIGCOMM*, 2011.

[30] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient Coflow Scheduling with Varys," in *Proc. ACM SIGCOMM*, 2014.

[31] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized Task-Aware Scheduling for Data Center Networks," in *Proc. ACM SIGCOMM*, 2014.

[32] A. Lester, Y. Tang, and T. Gyires, "Application-Aware Bandwidth Scheduling for Data Center Networks," *Journal on Advances in Networks and Services*, vol. 7, 2014.

[33] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," *RFC 2992*, 2000.

[34] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *Proc. ACM CoNEXT*, 2011.