Towards Application-aware In-network Bandwidth Management in Data Centers

Renhai Xu, Wenxin Li, Keqiu Li, Heng Qi
School of Computer Science and Technology, Dalian University of Technology, China
Keqiu Li is the corresponding author, keqiu@dlut.edu.cn

***Abstract*—A wide variety of applications nowadays are running on data centers, making the scarce in-network bandwidth of data centers become the performance bottleneck. To guarantee the network performance, an efficient in-network bandwidth management can be leveraged, however little work has been done so far. The crux is that existing work mainly relies on a static or dynamic bandwidth allocation strategy, which lacks efficient schemes to allow elastic and flexible use of the in-network bandwidth among different applications. In this paper, we study the in-network bandwidth management problem in data centers with a large number of diverse applications. To this end, we first formulate a max-min bandwidth constraint model to reserve an amount of bandwidth for each type of application. With this model, bandwidth can be sharing across different application types in an elastic and flexible way. We further present a Borrow-Return-Preempt bandwidth management method (BRP) to practically allocate bandwidth to individual requests of each type of applications. Extensive simulation results have shown that our proposed BRP is capable of improving the in-network bandwidth utilization as well as providing desirable network performance for applications, compared to the conventional fair allocation method.**

I. Introduction

Datacenter, implemented as the underlying infrastructure, has been increasingly popular in today's business. Using the shared pool of computation, storage and bandwidth resources, providers are able to deploy applications on their infrastructures. With the high-speed development of the Internet, business datacenters are increasingly hosting a wide variety of applications; from user-facing online services to data-parallel, HPC and scientific applications [1–6]. As a consequence, a large number of applications are sharing the intra-datacenter bandwidth, leading to a strong need for an efficient in-network bandwidth management scheme. Moreover, since many data centers are oversubscribed, i.e., as high as 40:1 in some datacenters [7], applications will suffer unpredictable performance if there is no bandwidth management.

Today, it is easy to share and manage computation and storage resources among multiple applications effectively. In contrast, designing an efficient bandwidth management method on a shared datacenter, however, is an inherently complex task, which has the following two challenges. First, the in-network bandwidth resources are comprised of bandwidth in a large number of switches, which are interconnected in a complicated manner [8, 9]). Moreover, the number of applications on a data center can be larger, and different application may have different bandwidth requirements. These all make it harder to manage the in-network bandwidth in modern data centers. Second, the amount of bandwidth actually consumed by an application can possibly vary over the time. Simply allocating the required bandwidth to applications would cause bandwidth fragmentation and waste because the bandwidth allocated to one application cannot be used by other applications, even if it is idle.

To the best of our knowledge, previous work on bandwidth allocation mainly relies on static allocation or dynamic allocation, yet has significant limitations in managing the in-network bandwidth in data centers. On the one hand, the static bandwidth allocation fairly allocates bandwidth to different entities. Such entities can be VM [10], VM-pair [11], or tenant [12]. Once the bandwidth is allocated, each entity will hold the bandwidth until its data transferring is completed. As a result, the intra-datacenter bandwidth cannot be efficiently utilized when entities are not able to fully utilize the bandwidth allocate to them. On the other hand, dynamic allocation is achieved by enforcing rate limit on traffic among endpoints, like VMs in virtualized datacenters [13, 14]. Unfortunately, since traffic is sent by the source, traversing through some intermediate (like, switches), and finally arrivals the destination, simply enforcing rate limit in the endpoints cannot capture the dynamic change in the intra-datacenter network. Furthermore, it will hurt the application performance, especially the performance for some latency-sensitive applications. The reason can be that it does not provide strict priority, even low priority applications, which should be paused, continue to send packets.

To efficiently manage the in-network bandwidth, there are more requirements that should be satisfied. First, the bandwidth management must be able to enforce different bandwidth constraints for different type of applications. As we known, the data centers have become the major platforms for the cloud computing, which is a multi-user environment. Hence, applications, submitted by a customer who pays more than other customers, should be provided with more bandwidth than others. Second, the bandwidth management must be able to process at least two parameters: the requested bandwidth and the priority. Processing requested bandwidth can bring benefit in providing desirable network performance for applications, while bandwidth preemption can ensure that a bandwidth request with high priority can get more bandwidth than that with low priority.

To satisfy these requirements, we first present a max-min bandwidth constraint model, where the reserved bandwidth of each application type is enforced between a minimal value and a maximum value. The idle part of the minimal reserved bandwidth for each application type is allowed to be borrowed by other application types, raising the opportunity to use the in-network bandwidth in an elastic and flexible way. Based on this model, we further present a Borrow-Return-Preempt (BRP) bandwidth management method to allocate bandwidth to the individual bandwidth requests of each application type. With the BRP method, the unused fraction of minimal reserved bandwidth of an application type can be borrowed by bandwidth requests from other types of applications. The bandwidth of an application type borrowed by other application types can be returned to this application type. Moreover, a high-priority bandwidth request of an application type is allowed to preempt the bandwidth from low-priority requests of both this application type and other application types. To evaluate the performance of our proposed BRP method, we conduct comprehensive simulations. The results have shown that BRP can improve the in-network bandwidth utilization by up to 19.7%, and accommodate 37% more requests with bandwidth guaranteed, compared to the conventional fair allocation method [15].

In summary, our main contributions are as follows:

1) We address the challenging problem of managing the in-network bandwidth across a mixed variety of applications in data centers. Specifically, we formulate a max-min bandwidth allocation model.
2) We present a Borrow-Return-Preempt method to allocate the in-network bandwidth among different applications in an elastic and flexible way.
3) We conduct comprehensive simulation to show the efficiency of our proposed BRP method, with respect to the performance on both the bandwidth utilization and request successive rate.

The rest of our paper is organized as follows. We present the problem statement and bandwidth constraint model in Section II. In Section III, we present the BRP method. Then, we show the performance evaluation in Section IV. Section V summarizes the related work. Finally, we conclude this paper in Section VI.

II. Problem Statement and Bandwidth Constraint Model

In this section, we first present some goals of our bandwidth management framework in a data center hosting a wide variety of applications. Then, we show the max-min bandwidth constraint model.

A. Problem Statement

Our focus in this paper is designing an efficient bandwidth management framework in a shared datacenter that hosts a mixed multiple applications. We aim to provide the in-network bandwidth guarantees for applications, and improve the bandwidth efficiency.

In-network bandwidth guarantees: As we know, shared by a wide variety of applications, the in-network bandwidth is usually the bottleneck resource, especially in a data center with high oversubscription ratio. To ensure predictable performance for applications, the requested bandwidth should be guaranteed. In this paper, we focus on the in-network bandwidth guarantee for applications. Such in-network bandwidth guarantee implies that the requested bandwidth should be guaranteed in all switches (or nodes) along the routing path of this request.

Efficiency: Traditionally, once an application acquired an amount of bandwidth, it will hold the bandwidth until its data transferring is completed. As a result, the bandwidth cannot be efficiently utilized when applications are not able to fully utilize the bandwidth allocated to them. Fortunately, as applications of different types have different bandwidth requirements, we can enforce the amount of bandwidth of each application type to be ranged from a minimum to a maximum value. Once a certain application type consumes an amount of bandwidth less than its minimum value, the leftover bandwidth should be lent to applications from other types. In case that it increases its demand, it must be able to preempt the bandwidth borrowed by others.

With the above two objectives, we consider a data center network with a wide variety of applications. We assume that each application can be differentiated, and can be classified in the corresponding application type. With the distributed bandwidth management, there are four main steps in establishing a new bandwidth request. First, we compute a shortest path for the coming request based on the network topology. Second, we send the requested bandwidth to all nodes along the computed path. Third, each node on the path maintains a bandwidth constraint model, and executes the Borrow-Return-Preempt bandwidth allocation. Finally, it sends a positive reply to the source. Forth, if all the nodes along the path return positive replies, then the flow establishing is successful, and the requested bandwidth will be reserved on all nodes.

B. Max-min Bandwidth Constraint Model

As aforementioned, each node in the data center network maintains a bandwidth constraint model. We consider that each node i has a bandwidth capacity U_i and mainly maintains two matrices, C_i and A_i. The matrix C_i is an $N_{at} \times N_p$ matrix, which is used to record bandwidth consumption, with its element $C_i[k, l]$ being the bandwidth consumed by application type k at priority l. N_{at} is the number of application types, while N_p is the number of priority levels. Similarly, A_i is used to record the bandwidth availability, with each element $A_i[k, l]$ being the available bandwidth to application type k at priority l.

Recall that we aim to design a dynamic bandwidth management framework with high bandwidth efficiency. Moreover, the unused bandwidth of an application type should be used for other application types. In order to enhance the efficiency, we let the amount of bandwidth used for a certain application type be ranged from a minimum to a maximum value. More precisely, $R_i^{min}[k]$ and $R_i^{max}[k]$ are the corresponding minimum and maximum amount of bandwidth reserved for application type k on node i, respectively. Let $B_i[k_1, l, k]$ denote the amount of bandwidth, belonging to application type k, that borrowed by requests from application type k_1 with priority l. Let $T_i[k, l]$ denote the fraction of minimal reserved bandwidth that consumed by requests of application type k with priority l. Note that the total amount of bandwidth that is sharable among all applications can be calculated as $U_i - \sum_{k=0}^{N_{at}-1} R_i^{min}[k]$. So, in the sharable bandwidth, let $S_i[k, l]$ denote the bandwidth consumed by application k with priority l. The total amount of bandwidth consumed by application type k with priority l can now be calculated as

$$C_i[k, l] = T_i[k, l] + S_i[k, l] + \sum_{k_1 \neq k}^{N_{at}-1} B_i[k, l, k_1]. \tag{1}$$

Now, we have the following bandwidth constraint model.

$$\sum_{l=0}^{N_p-1} C_i[k, l] \leq R_i^{max}[k], \ 0 \leq k \leq N_{at} - 1 \tag{2}$$

$$R_i^{min}[k] \leq R_i^{max}[k], \ 0 \leq k \leq N_{at} - 1 \tag{3}$$

$$\sum_{k=0}^{N_{at}-1} R_i^{min}[k] \leq U_i, \tag{4}$$

$$\sum_{k=0}^{N_{at}-1} \sum_{l=0}^{N_p-1} C_i[k, l] \leq U_i, \tag{5}$$

$$\sum_{l=0}^{N_p-1} T_i[k, l] + \sum_{k_1 \neq k}^{N_{at}-1} \sum_{l=0}^{N_p-1} B_i[k_1, l, k] \leq R_i^{min}[k], \tag{6}$$

$$\sum_{k=0}^{N_p-1} \sum_{l=0}^{N_p-1} S_i[k, l] \leq U_i - \sum_{k=0}^{N_{at}-1} R_i^{min}[k]. \tag{7}$$

Eq. (2) implies that each application type k has a maximum amount of bandwidth consumption $R_i^{max}[k]$. As shown in Eq. (3), the minimum amount of bandwidth reserved for one application type must be less than its corresponding maximum value. In order to avoid congestion, the sum of the minimum amount of reserved bandwidth should not exceed the bandwidth capacity U_i, as shown in Eq. (4). Eq. (5) means that the total amount of actually consumed bandwidth should not exceed the bandwidth capacity. Eq. (6) shows that the sum of T_i and B_i should not exceed the minimum amount of reserved bandwidth. In the part of shareable bandwidth, the bandwidth consumed by application type k with priority l should not exceed the amount of sharable, as shown in Eq. (7). It is clear that this model enforces the amount of reserved bandwidth for each application type to be ranged from a minimal value to a maximum value, which accounts for the max-min bandwidth constraint model.

1) Update the bandwidth consumption matrix: Once a certain amount of bandwidth bw on node i is allocated to or released from application type at at priority level p, the matrix C_i can then be updated as follow

$$C_i[k, l] = C_i[k, l] + bw, \tag{8}$$

where bw has positive value for a bandwidth grant and a negative value for a bandwidth release.

2) Update the bandwidth availability matrix: Based on the bandwidth constraint model and the computed bandwidth consumption matrix, the available bandwidth of application type k can be computed as the following equation

$$A_i[at, p] = \min\{R_i^{max}[at] - \sum_{l=0}^{p} C_i[at, l], g_{return\_at} + g_{min\_at} + g_{share} + g_{borrow\_at} + g_{prt\_at} + g_{prt\_ot}\}, \tag{9}$$

where

$$g_{return\_at} = \sum_{k \neq at}^{N_{at}-1} \sum_{l=0}^{N_p-1} B_i[k, l, at], \tag{10}$$

$$g_{min\_at} = R_i^{min}[at] - \sum_{l=0}^{N_p-1} T_i[at, l] - g_{return\_at}, \tag{11}$$

$$g_{share} = U_i - \sum_{k=0}^{N_{at}-1} R_i^{min}[k] - \sum_{k=0}^{N_{at}-1} \sum_{l=0}^{N_p-1} S_i[k, l], \tag{12}$$

$$g_{borrow\_at} = \sum_{k \neq at}^{N_{at}-1} g_{min\_k}, \tag{13}$$

$$g_{prt\_at} = \sum_{l=p+1}^{N_p-1} C_i[at, l], \tag{14}$$

$$g_{prt\_ot} = \sum_{k \neq at}^{N_{at}-1} \sum_{l=p+1}^{N_p-1} (C_i[k, l] - T_i[k, l]). \tag{15}$$

In Eq. (9), $R_i^{max}[at] - \sum_{l=0}^{p} C_i[at, l]$ computes the portion of the maximum bandwidth that is able to the new flows of application type at with priority p. Additionally, the available bandwidth for the flows of application type at with priority p consists of 6 parts: 1) g_{return_at}, the portion of minimal reserved bandwidth that application type at previously lent to other application types, which should be returned once application type at increases its demand (Eq. 10); 2) g_{min_at}, the portion of minimum reserved bandwidth which is not consumed (Eq. (11)); 3) g_{share}, the leftover sharable bandwidth (Eq. (12)); 4) g_{borrow_at}, the total unconsumed minimum reserved bandwidth that can be borrowed by application type at (Eq. (13)); 5) g_{prt_at}, the total amount of consumed bandwidth by application type at's flows with priority lower than p (Eq. (14)); 6) g_{prt_ot}, defined as the total amount of consumed bandwidth minus the total amount of consumed minimum reserved bandwidth, associated with flows of all application types except at with priority lower than p (Eq. (15)).

III. Borrow-Return-Preempt Bandwidth Management Model

In this section, we present a bandwidth-return-preempt bandwidth management model to allocate bandwidth to the requests issued by a mixed variety of applications.

Algorithm 1 Borrow Algorithm: at type application borrow bandwidth from k type application on node i

```
Input: $b\_bw, g_{borrow\_at}, g_{min\_k}, g_{return\_k}, B_i$;
Output: A new $b\_bw$;
1: $g_{borrow\_at} -= b\_bw$;
2: while $b\_bw > 0$ do
3:     Search a Application type $k (k \neq at)$ with the biggest $g_{min\_k} > 0$
4:     $r[np] = \min\{g_{min\_k}, b\_bw\}; b\_bw -= r[np]$;
5:     $g_{min\_k} -= r[np]; g_{return\_k} += r[np]$;
6:     $B_i[at, p, k] += r[np]; np{+}{+}$;
7: end while
8: return $b\_bw$;
```

Algorithm 2 Return Algorithm: k type application return bandwidth to at type application on node i

```
Input: $r\_bw, g_{return\_at}, T_i, B_i, C_i$;
Output: A new $r\_bw$;
1: $g_{return\_at} -= r\_bw$;
2: $T_i[at, p] += r\_bw$;
3: while $r\_bw > 0$ do
4:     Search a non-zero item $B_i[k, l, at] (k \neq at)$ with the biggest $l \leq N_p - 1$
5:     $r[np] = \min\{B_i[k, l, at], r\_bw\}; r\_bw -= r[np]$;
6:     $B_i[k, l, at] -= r[np]; C_i[k, l] -= r[np]; np{+}{+}$;
7: end while
8: return $r\_bw$;
```

A. Design Fundamental

We first present some fundamentals for our bandwidth management. Once a node receives a request (bw, at, p), it will invokes the bandwidth allocation for this request. To ease the presentation, let the unconsumed bandwidth of application type at at the arrival time of the new request (bw, at, p) be defined by

$$\begin{aligned} unconsumed_i[at] = \min\{&U_i - \sum_{k=0}^{N_{at}-1} \sum_{l=0}^{N_p-1} C_i[k, l], \\ &R_i^{max}[at] - \sum_{l=0}^{N_p-1} C_i[at, l] - g_{return\_at}\}, \end{aligned} \tag{16}$$

which implies that a new request (bw, at, p) can be accommodated with any unconsumed portion of the bandwidth capacity on the node; that is $U_i - \sum_{k=0}^{N_{at}-1} \sum_{l=0}^{N_p-1} C_i[k, l]$. Besides, the accommodation of the new request should not cause application type at's consumption to exceed $R_i^{max}[at]$.

Given the $unconsumed_i[at]$, the node can perform bandwidth allocation for the coming request (bw, at, p). It first checks whether the requested bandwidth bw is less than or equal to the available bandwidth $A_i[at, p]$. If yes, our algorithm computes the unconsumed bandwidth $unconsumed_i[at]$ based on Eq. (16). The unconsumed bandwidth for application type at may contain the unused minimum reserved bandwidth, the sharable bandwidth, and the bandwidth that previously borrowed by other application types. So, the key idea in designing the bandwidth allocation on each node is to make decisions on bandwidth borrowing, bandwidth returning and bandwidth preemption.

B. Borrow-Return-Preempt Allocation

We seamlessly combine the following three algorithms, and design a BRP method to allocate bandwidth to each coming request.

Borrow algorithm: If the bandwidth request bw exceed the sum of g_{min_at} and g_{share}, this means that the g_{min_at} and g_{share} cannot satisfy the bandwidth requirement. It is necessary to borrow bandwidth from other application types, as shown in Algorithm 1. The b_bw is the amount of bandwidth that application at needs to borrow. Step 3 means that each time we borrow bandwidth from application k with the biggest g_{min_k} which is the unconsumed bandwidth in the minimum reserved bandwidth $R^{min}[k]$. Finally, update the $B_i[at, p, k]$ and g_{min_k}.

Return algorithm: If bw is bigger than $unconsumed_i[at]$ but less than or equal to the sum of $unconsumed_i[at]$ and g_{return_at}, the unconsumed bandwidth is insufficient. Hence, the application type at with priority p should retrieve bandwidth which has been borrowed by application type $k (k \neq at)$. As shown in Algorithm 2, the parameter r_bw is the amount of bandwidth that other applications should return to application at. The algorithm searches a non-zero item $B_i[k, l, at] (k \neq at)$ with the biggest $l \leq N_p - 1$, until r_bw reduces to 0. Finally, it updates the g_{return_at}, $T_i[at, p]$ and $B_i[k, l, at]$.

Preemption algorithm: The node may discover that the requested bandwidth is less than or equal to the available bandwidth $A_i[at, p]$, but there is inadequate bandwidth to accommodate the new request because a portion of the available bandwidth has been occupied by the existing flows with priorities lower than p. In this case, the bandwidth preemption will be invoked. The preemption process is summarized in Algorithm 3. The preemption decisions are made based on both the priority level and the bandwidth consumption status, so as to minimize the amount of bandwidth preempted. As shown in step 1-13, to accommodate this request, this algorithm first preempts bandwidth from application type at with priority lower than p. If preempted bandwidth still cannot satisfy the requested bandwidth, then this algorithm preempts bandwidth from other application type k with priority lower than p, as shown in step 14-24. It should be noted that a necessary condition when invoking the Algorithm 3 should be satisfied: the requested bandwidth must be larger than the summation of $unconsumed_i[at]$ and g_{return_at}.

Finally, the bandwidth consumption matrix C_i and the bandwidth availability matrix A_i should be updated, and a flag should also be returned to indicate whether the request is successfully served on node i.

Algorithm 3 Preemption Algorithm: preempt application with priority $l (l > p)$ on node i

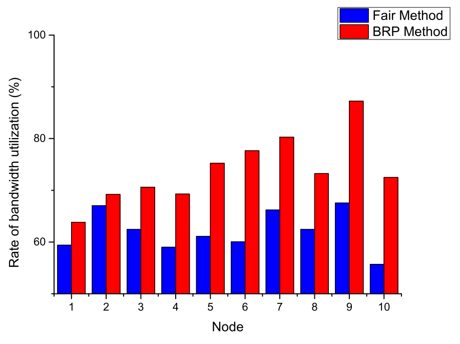
```
Input: $p\_bw, C_i, T_i, S_i, B_i$;
Output: A new $p\_bw$;
 1: while $p\_bw > 0$ do
 2:     Search a non-zero item $C_i[at, l]$ with the biggest $l > p$
 3:     $p[np] = \min(T_i[at, l], p\_bw)$;
 4:     update $p\_bw$, $T_i[at, l]$ and $T_i[at, p]$;
 5:     $p[np] = \min(S_i[at, l], p\_bw)$;
 6:     update $p\_bw$, $S_i[at, l]$ and $S_i[at, p]$;
 7:     while $p\_bw > 0$ do
 8:         Search a non-zero item $B_i[at, l, k]$ with $k \neq at$
 9:         $p[np] = \min(B_i[at, l, k], p\_bw)$;
10:         update $p\_bw$, $B_i[at, l, k]$ and $B_i[at, p, k]$;
11:     end while
12:     $np{+}{+}$;
13: end while
14: while $p\_bw > 0$ do
15:     Search a non-zero item $C_i[k, l]$ with the biggest $l > p$
16:     $p[np] = \min(S_i[k, l], p\_bw)$;
17:     update $p\_bw$, $S_i[k, l]$ and $S_i[at, p]$;
18:     while $p\_bw > 0$ do
19:         Search a non-zero item $B_i[k, l, k_1]$ with $(k_1 \neq at)$
20:         $p[np] = \min(B_i[k, l, k_1], p\_bw)$;
21:         update $p\_bw$, $B_i[at, p, k_1]$ and $B_i[k, l, k_1]$;
22:     end while
23:     $np{+}{+}$;
24: end while
25: return $p\_bw$;
```

IV. Performance Evaluation

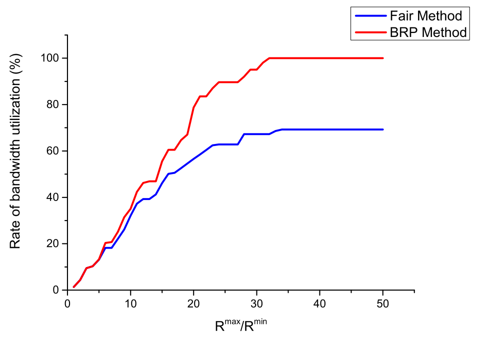
In this section, we conduct comprehensive simulations to evaluate our proposed BRP method.

A. Experiment Setting

To evaluate the performance of our Borrow-Return-Preempt bandwidth allocation method, we use a 10-node Fat-Tree topology. Without loss of generality, the bandwidth capacity of each node is uniformly set to 2Gbps. We consider 10 types of applications, e.g., $N_{at} = 10$, and the total number of priority levels is set to 100 ($N_p = 100$). For a certain period of time, there are $N_{at} \times N_p$ flows in total. Each flow has an amount of bandwidth ranging from 0Mbps to 100Mbps. The minimal reserved bandwidth configured for each application type is randomly selected in the range of $[0, 50]$Mbps. For the maximum reserved bandwidth, we vary it based on the minimum reserved bandwidth. More previously, if the minimum reserved bandwidth for a node is 10Mbps, then the corresponding maximum reserved bandwidth is set to $1\times, 2\times, \cdots, 50 \times$ 10Mbps. So, we actually conduct 50 runs in our experiments. We compared BRP with the default fair bandwidth allocation method, which allocates a same amount of bandwidth to each active flow on a node [15]. It should be noted that we also maintains the max-min bandwidth constraint model for the fair bandwidth allocation



(a) Bandwidth utilization on each node



(b) Bandwidth utilization with varying R^{max}/R^{min}

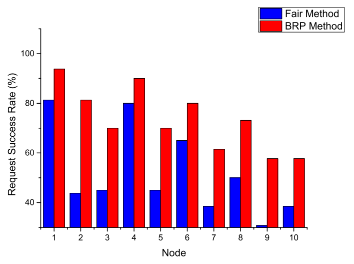
Fig. 1. The performance on bandwidth utilization

in our experiments. We focus on evaluating the following two metrics: the bandwidth utilization and the request success rate.

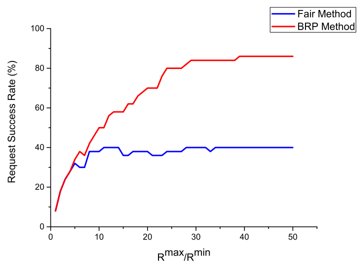
B. Bandwidth Utilization

We first evaluate the performance on the bandwidth utilization. Fig. 1(a) first presents the bandwidth utilization across all nodes. It is clear that across all nodes, our BRP method achieves a higher bandwidth utilization rate than the fair allocation method. More previously, the BRP can improve the bandwidth utilization by up to 19.7%, compared to fair allocation method. The average improvement on the bandwidth utilization is 11.8%. The root reason is that though we maintain the max-min bandwidth constraint model for the fair allocation method, the unused minimal reserved bandwidth of each application type cannot be used for other application types, even it is idle. This confirms that our BRP enables elastic and flexible bandwidth use among a mixed variety of applications.

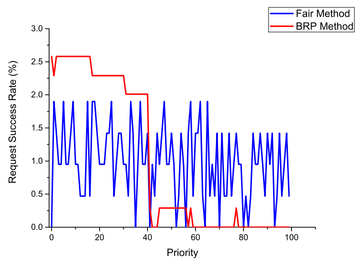
To evaluate the impact of R^{max}/R^{min} on the bandwidth utilization, we plot the bandwidth utilization for both BRP and fair allocation methods, with varying values of R^{max}/R^{min} in Fig. 1(b). As we can see, the bandwidth utilization of both BRP and fair allocation methods increases at the beginning, and then maintains a stable value when R^{max}/R^{min} increases to 35. Moreover, BRP can fully utilize the bandwidth when



(a) Request success rate across all nodes



(b) Request success rate with varying R^{max}/R^{min}



(c) Request success rate at different priority levels

Fig. 2. The performance on request success rate

R^{max}/R^{min} increases to 35, while fair allocation method can only utilize 69.3% of bandwidth. This result again verifies that our BRP method can allocate bandwidth in a dynamic way, raising the opportunity to fully utilize the bandwidth across all nodes.

C. Request success rate

We now evaluate the performance on the request success rate for both BRP and fair allocation methods. Fig. 2(a) first plots the request success rate across all nodes. It is clear that the BRP method can achieve a higher request success rate across all nodes. More preciously, across all nodes, the maximum improvement in the request success rate is 37%. Averagely, BRP can accommodate 21.9% more requests the fair allocation method across all nodes. The crux is that BRP can use the leftover part of the minimal reserved bandwidth for each application type. We also conduct sensitivity analysis on the request success rate by varying the value of R^{max}/R^{min}. As shown in Fig. 2(b), the request success rate increases as the increase of R^{max}/R^{min}. With a sufficient large value of R^{max}/R^{min}, e.g., 40, BRP can accommodate 86% requests, while fair allocation method only accommodates 40% requests. To understand on a microcosmic level, we also conduct quantitatively analysis on the success rate for requests at different priority levels. As show in Fig. 2(c), BRP method ensures that requests with high-priority are more likely to be successfully served than requests with lower priorities. However, fair allocation seems to achieve a request success rate irrespective to the priorities of different requests. This is because that fair allocation method equally allocates bandwidth to the concurrent requests. Given these results, we can conclude that our proposed BRP method can achieve a higher request success rate than the default bandwidth sharing method.

V. Related Work

In this section, we mainly summarize the related work on bandwidth allocation in data centers, as this is mostly related to our work. Existing methods on bandwidth allocation can be generally classified into two folds: static bandwidth allocation and dynamic bandwidth allocation. However, none of them can address the problem of managing the in-network bandwidth in a data center with a mixed variety of applications.

Regarding the static bandwidth allocation, some methods focus on reserving bandwidth during the virtual machine (VM) placement, while some methods focus on the bandwidth allocation after VM placement. For example, SecondNet [16] reserves bandwidth for each VM-to-VM pari. Lee et al. propose a new network abstraction TAG (tenant application graph) and reserve bandwidth at the application level [17]. Zhu et al. [18] focus on the problem of VM allocation under the consideration of providing bandwidth guarantees with both homogeneous and heterogeneous bandwidth demand considered. For allocating bandwidth after the VM placement, existing methods mainly focus on fairly allocate bandwidth to different entities. For instance, Faircloud presents three methods for allocating bandwidth on congested links, which can achieve the VM-pair fairness [11]. Guo et al. propose an allocation strategy based on game theory, which keeps fairness among different VMs [10]. NetShare [12] provides tenant-level fairness on congested links and achieves proportional bandwidth sharing by using weighted fair queues. Chen et al. focus on application-level fairness and they introduce a rigorous definition of performance-centric fairness with the guiding principle that the performance of data parallel applications should be proportional to their weights [19]. These methods mainly focus on static bandwidth allocation with ignoring the highly dynamic bandwidth request of VMs, leading to insufficient in utilizing the bandwidth in a flexible and elastic way.

Regarding the dynamic bandwidth allocation, existing methods mainly enforcing rate limit on the endpoints such as servers and VMs. For example, Popa et al. present ElasticSwitch to dynamic utilize the spare bandwidth to the newly coming flows [13]. It can be fully implemented in hypervisors, but it has fluctuations under bursty traffic when the rate limit is beyond the guarantee. Guo et al. take advantage of the Logistic Model to design a novel distributed bandwidth allocation algorithm, with the aim of coping with highly dynamic traffic in the datacenter network [14]. To perform bandwidth allocation, most of them rely on a technique of rate limit on the endpoints. However, such rate limit has significant limitations: since each network flow traverses through multiple switches among its routing path, such rate limit is unaware of the in-network status of data centers. This eventually hurts the performance of applications.

VI. Conclusion

This paper studies the challenging problem of managing the in-network bandwidth in data centers multiplexed with a mixed variety of applications. The primary objective is to improve the bandwidth utilization while providing desirable network performance for applications. To achieve this objective, we first present a max-min bandwidth constraint model, which enforces a minimal as well as a maximum amount of bandwidth reserved for each type of application. Then, we present a Borrow-Return-Preempt method to allocate bandwidth to the individual bandwidth requests of each type of application. Finally, we conduct comprehensive simulations to show the efficiency of our proposed BRP method in terms of both the bandwidth utilization and the request success rate.

Acknowledge

This work is supported by the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61225010); the State Key Program of National Natural Science of China (Grant 61432002).

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] Q. He, S. Zhou, B. Kobler, D. C. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," in *Proceedings of ACM HPDC*, Chicago, Illinois, USA, 2010.

[3] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," *USENIX Login*, vol. 33, no. 5, pp. 18–23, 2008.

[4] "Storm: Distributed and fault-tolerant realtime computation," http://storm-project.net/.

[5] "Spark," http://spark.apache.org/.

[6] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Prooceedings of ACM EuroSys*, 2007.

[7] N. Farrington and A. Andreyev, "Facebooks data center network architecture," in *Proceedings of IEEE Optical Interconnects*, Santa Fe, NM, 2013.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of ACM SIGCOMM*, 2008.

[9] 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 *Proceedings of ACM SIGCOMM*, 2009.

[10] J. Guo, F. Liu, D. Zeng, J. Lui, and H. Jin, "A cooperative game based allocation for sharing data center networks," in *Proceedings of IEEE INFOCOM*, Turin, Italy, 2013.

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

[12] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "Netshare: Virtualizing data center networks across services," University of California, San Diego, Tech. Rep., 2010.

[13] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing," in *Proceedings of ACM SIGCOMM*, 2013.

[14] J. Guo, F. Liu, X. Huang, J. C.S.Lui, M. Hu, Q. Gao, and H. Jin, "On efficient bandwidth allocation for traffic variability in datacenters," in *Proceedings of IEEE INFOCOM*, Toronto, Canada, 2014.

[15] B. Briscoe, "Flow rate fairness: Dismantling a religion," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 63–74, 2007.

[16] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the ACM CoNEXT*, Philadelphia, PA, USA, 2010.

[17] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *Proceedings of ACM SIGCOM*, Chicago, USA, 2014.

[18] J. Zhu, D. Li, J. Wu, H. Liu, Y. Zhang, and J. Zhang, "Towards bandwidth guarantee in multi-tenancy cloud computing networks," in *Proceedings of IEEE ICNP*, Austin, TX, USA, 2012.

[19] L. Chen, Y. Feng, B. Li, and B. Li, "Towards performance-centric fairness in datacenter networks," in *Proceedings of IEEE INFOCOM*, Toronto, Canada, 2014.