symmetry

Article

Bandwidth-Guaranteed Resource Allocation and Scheduling for Parallel Jobs in Cloud Data Center

Zhen Li [1], Bin Chen [1,*], Xiaocheng Liu [1], Dandan Ning [1], Qihang Wei [2], Yiping Wang [3] and Xiaogang Qiu [1]

[1] College of System Engineering, National University of Defense Technology, Changsha 410073, China; nudt.lz1991@gmail.com (Z.L.); nudt200203012007xcl@gmail.com (X.L.); nudtndd@163.com (D.N.); michael.qiu@139.com (X.Q.)
[2] The 66029th Troop of PLA, Inner Mongolia Autonomous Region 011216, China; wqh85563028@163.com
[3] The Naval 902 Factory, Shanghai 200083, China; foolwangrain@126.com
* Correspondence: nudtcb9372@gmail.com; Tel.: +86-0731-84574332

Received: 29 March 2018; Accepted: 19 April 2018; Published: 25 April 2018

Abstract: Cloud Computing has emerged as a powerful and promising way for running high performance computing (HPC) jobs. Most HPC jobs are designed under multi-processes paradigm and involve frequent communication and synchronization among parallel processes. However, as the underlying resources of cloud data centers are always shared among multiple tenants, the competition of jobs for limited bandwidth resources lead to unpredictable completion times for jobs in the cloud, which may lead to QoS violation and inefficient utilization of resources when scheduling parallel jobs in the cloud. To tackle the issue, it is essential to provide bandwidth guarantees for parallel jobs running in the cloud. Offering a dedicated virtual cluster (VC) for running applications in the cloud is a popular way to guarantee bandwidth demands. Motivated by these problems, in this paper, we firstly design a time-aware virtual cluster (TVC) request model for parallel jobs and consider how to embed requested TVCs of jobs into cloud efficiently under parallel job scheduling framework. An adaptive bandwidth-aware heuristic algorithm, which is denoted as AdaBa, is proposed to improve the job accept rate by adjusting the priorities of servers to accommodate the VMs of TVC adaptively according to the relative size of requested bandwidth demand. Then, a bandwidth-guaranteed migration and backfilling scheduling algorithm, which is denoted as BgMBF, is designed to schedule parallel jobs and the bandwidth demands are guaranteed by AdaBa. To obtain high job responsiveness performance, a bandwidth-reserved job backfilling strategy is designed when the requested TVC for current scheduled job cannot be allocated in the cloud. The migration cost of BgMBF is also considered and an enhanced version BgMBFSDF is then proposed to minimize the number of migration when the execution time of jobs are known. Through extensive simulation experiments on popular parallel workloads, our proposed TVC embedding algorithm AdaBa achieves up to 15 percent of improvement on accept rate compared with existing algorithms such as Oktupus and greedy algorithm. Our proposed BgMBF and BgMBFSDF also significantly outperform other popular scheduling algorithms integrated with AdaBa on average response time and average bounded slow down.

Keywords: virtual cluster embedding; parallel job scheduling; cloud data center; bandwidth allocation

1. Introduction

Based on virtualization, data management techniques, etc., cloud computing paradigm delivers not only cost-effective and powerful Infrastructure as a Service (IaaS), but also flexible and customized Platform as a Service (PaaS) and Software as a Service (SaaS), which allow agile customization to specific applications, software, and programming environment needs of users. More and more HPC users are being attracted to request cloud computing service instead of owning a cluster for running high performance computing (HPC) applications. Cloud providers, such as Amazon EC2, have also made efforts on supporting scalable running of HPC applications [1].

However, as the requirements of HPC applications are mismatched with the characteristics of cloud natively [2–4], resources allocation and job scheduling for HPC applications in the cloud are still popular and challenging topics [5]. Two main factors that contribute to the problem can be summed up as follows:

1. HPC applications are typically designed under multi-processes paradigm and consist of tightly-coupled parallel processes with frequent communication and synchronization. Insufficient network bandwidth resources and large network latency lead to performance degradation for HPC in cloud compared with that in a local dedicated cluster.
2. Variable data transmission latency caused by bandwidth resource contention under shared and multi-tenant cloud environment leads to unpredictable performance for parallel jobs of HPC. The performance uncertainty also results in poor job scheduling performance and low throughput.

Accordingly, HPC applications can be considered as time-critical applications which may likewise involve some MPI or other parallel computing based components for high performance data processing [6]. Recent studies have made efforts on the supporting tools for the development, management and control of time-critical cloud applications [7]. However, there are still few studies in job scheduling considering QoS guarantee for HPC applications in the cloud. Recently, bandwidth-guaranteed virtual cluster embedding techniques have been extensively studied. It offers a virtual dedicated network environment for applications in the cloud and leads to predictable performance by providing resource guarantees including bandwidth, CPU and memory. Different virtual cluster abstraction models are proposed to accurately specify resource demands according to the characteristics of submitted applications, such as Pipe [8], VC [9], SVC [10], etc. However, the resource allocation for virtual cluster is a NP problem which is known as the virtual cluster embedding problem. Many heuristic algorithms are accordingly proposed. Oktopus [9] introduces a virtual cluster model named VC, and designs a locality-aware heuristic algorithm which tries to allocate the requested VMs of VC to servers by minimizing the number of switches on the communication path. Through the locality-aware allocation strategy, maximum bandwidth reservation for network links at higher levels of substrate network topology can be obtained, and more future VC requests are expected to be accepted. Another greedy strategy is to minimize the maximum link occupation [10,11]. This strategy tries to assign the VMs to the servers that leads to the minimum value of maximum link occupation [11]. To improve the accept rate of jobs, a congestion-aware perturbation algorithm is also proposed which selectively relocates some assigned VMs to avoid the bandwidth bottleneck [12]. There are also many other enhanced algorithms which consider including the heterogeneity [13], uncertainty [10], time variance [14], etc. of the bandwidth demands of the links in the virtual cluster.

Nevertheless, these algorithms only focus on improving the acceptance rate of current arrived jobs, and seldom consider the bandwidth demand of the following arriving jobs. However, the bandwidth demands of parallel jobs vary in a wide range, from embarrassing parallel with little communication to large scale data-intensive parallel. Accordingly, the virtual cluster embedding algorithm for parallel jobs should be qualified with adaptive resource allocation according to the arrived bandwidth demand to maximize the residual bandwidth capabilities that can be used for the following waiting jobs after embedding current virtual cluster. On the other hand, the huge bandwidth resources of loopback network in each server are seldom considered in the literature, which provides chances to accommodate extra jobs with large bandwidth demands.

Considering the parallel job scheduler, the most basic but popular scheduling algorithm is first come first serve (FCFS). The scheduler dispatches jobs according to the order of their arrivals, and no jobs can be dispatched until the first job in the queue is allocated. FCFS is easy but may cause resource fragmentation and methods such as job backfilling [15], and gang scheduling [16] is proposed to improve it. However, current batch scheduling algorithms for parallel jobs are mainly designed for the HPC clusters. Bandwidth guarantees are not considered as the bandwidth resources are sufficient under the LAN environment of cluster. Liu [17] proposed a priority-based consolidation scheduling method for parallel jobs in the cloud, but it mainly focuses on improving the utilization degradation of cloud data center caused by communication and synchronization of parallel jobs. Dalvandi [14] proposed a time-aware virtual cluster request model which can be used to specify an estimated required time-duration for jobs, and designed several online heuristic algorithms to allocate resources for scheduled requests. To improve the accept rate of jobs, the scheduling algorithm tries to pick the most suitable requested jobs to execute in the cloud according to the bandwidth and time duration profiles. It also cannot be directly used for parallel job scheduling, but the time-aware virtual cluster request model inspires us to model the parallel job request.

In the paper, we focus on scheduling parallel jobs running in the cloud with predictable performance and high responsiveness performance. To deal with the issue, a bandwidth-guaranteed parallel job scheduling framework is proposed. The framework consist of two main parts: virtual cluster embedding component and parallel job scheduler.

The virtual cluster embedding component is responsible for allocating the bandwidth and VM resources requested by scheduled jobs. Inspired by Dalvandi et al. [14], we design a time-aware virtual cluster (TVC) request model to specific the resource demands of parallel jobs. Different from other proposed virtual cluster embedding algorithms, we designed an adaptive bandwidth-aware (AdaBa) TVC allocation algorithm which changes the embedding strategy of VMs based on both current bandwidth demand and maximum bandwidth demand that can be arrived adaptively. The basic idea of AdaBa is to hide bandwidth occupation of TVC with high bandwidth demand by allocating the VMs requested into the same server. Through the method, the jobs with high bandwidth demand are expected to communicate through loopback network inside the server, and more bandwidth resources on the network are gained to accept more jobs in the waiting queue. At the same time, the jobs whose parallel processes are located in the same host can achieve higher and more steady performance than that in separated hosts.

For parallel job scheduler, we design a bandwidth-guaranteed migration and backfilling scheduling algorithm (BgMBF) to schedule parallel jobs in the cloud. Our BgMBF extends existing AMBF algorithm [17] by integrating AdaBa algorithm to guarantee requested bandwidth demand while guaranteeing fairness (the jobs should not be delayed by the jobs arrived later) among the jobs. An optimized version named BgMBFSDF is also devised to improve the scheduling performance by choosing backfilled jobs with small execution duration first strategy to maximize the number of successful backfilled jobs that finish before being preempted.

In summary, the contributions of this paper are shown as follows:

1. We propose an efficient adaptive bandwidth-aware virtual cluster embedding algorithm to allocate requested resources of virtual cluster for scheduled parallel jobs running in the cloud, which excavates more bandwidth resources on the links through adaptive communication hidden strategy to improve the accept rate for following arriving jobs.
2. We design a BgMBF algorithm which backfills virtual clusters of waiting jobs to the idle fragmentation resources, to improve the average responsiveness performance. An enhanced algorithm of BgMBFSDF is also proposed to improve the scheduling performance by maximizing the number of successful backfilled jobs that finish before being preempted.
3. We demonstrate the efficiency of AdaBa and BgAMBF algorithms by extensive simulations. The results show that our proposed AdaBa can achieve higher acceptance rate compared with popular VC allocation algorithms under different workloads. The proposed BgMBF and enhanced BgMBFSDF algorithm significantly outperform FCFS and other heuristic scheduling strategies.

The remainder of this paper is organized as follows: Section 2 discusses some related work. Section 3 presents the system model and gives the scheduling framework along with problem definition of parallel jobs scheduling in the cloud. Detailed descriptions of our proposed algorithms are given in Section 4. Section 5 evaluates the performance of our algorithms. Section 6 concludes the paper and discusses future work.

2. Related Work

2.1. Bandwidth Allocation in the Cloud

Link bandwidth allocation has become the most critical issue to offer steady and predictable network performance in cloud data centers. Much research attention has been drawn to the issue in both academia and industry. Generally, the studies from the literature can be divided into two different allocation strategies. The first strategy is to provide bandwidth guarantees through static bandwidth reservations for each competing entity according to its demand. The advantage is that deterministic network performance can be obtained. However, the disadvantage is also evident: it may lead to low bandwidth utilization considering the time-varying traffic demands of tenant jobs. The second strategy is to to share bandwidth in a weighted sharing fashion among all tenants [18–21]. It can make full utilization of the bandwidth resources, but the bandwidth requirement for each tenant cannot be guaranteed. Accordingly, it can hardly satisfy the QoS of tenants when scheduling parallel jobs.

To accurately specify the static bandwidth reservations for different kinds of applications, several virtual network abstractions [8–10,14,22] have been proposed. SecondNet [8] designs a Pipe model which enables end-to-end bandwidth guarantee for VMs. However, in most cases, it is hard for the tenants to figure out each end-to-end bandwidth demand of applications, and it also introduces extra complexity when scheduling the jobs. In contrast, hose model simulates the cluster network where all VMs are connected to a central virtual switch by dedicated links (as hoses) with bandwidth guarantees. Based on the basic hose model, Oktopus [9] proposes the abstraction of virtual cluster (VC), which is shown as a 2-tuple $< N, B >$. It specifies N VM slots in the virtual cluster and each VM is connected by link of bandwidth B to the virtual switch. Locality-aware VM allocation algorithm is proposed in Oktopus to map the virtual units in VC abstraction to the physical facility. Yu [23] studied the survivable VC embedding problem with hose model bandwidth guarantee, which focuses on minimizing VM consumption for providing survivability guarantee. Furthermore, Yu [10] extended the VC abstraction with stochastic bandwidth characterization and proposed a stochastic virtual cluster (SVC) model to address the bandwidth demand uncertainty. As the bandwidth demands are given in a probabilistic way, probabilistic bandwidth guarantee are introduced which means that bandwidth demands are guaranteed with a high probability exceeding a specific threshold. Based on SVC, an efficient greedy VM allocation algorithm is then devised to reduce the possibility of link congestion through minimizing the maximum link occupancy when allocating requested VMs of virtual cluster. Dalvandi [14] proposed a novel time-aware request model which enables tenants to specify an estimated required time-duration in addition to CPU and bandwidth requirement, which inspired us to model the parallel job request. However, the time-aware request model is designed based on flow-based request model which needs to specify each end-to-end bandwidth requirement. DCloud [22] is another extension of the VC model, which requires a tenant to specify both the required resources and deadline of jobs. Without avoiding the deadline constraint of jobs, DCloud employs both time sliding (postponing the launching time of a job) and bandwidth scaling (adjusting the bandwidth associated with VMs) strategies in resource allocation algorithm, so as to better match the resource allocated to the job with the cloud's residual resource. However, the efficiency of DCloud highly depends on accurate estimation of job's execution time when carrying out bandwidth scaling.

2.2. Parallel Job Scheduling

In the literature, many efforts have been made on batch scheduling for parallel jobs in clusters [24]. The basic batch scheduling algorithm is First-Come-First-Serve (FCFS) [25]. Under this algorithm, each job specifies the number of processors required and the scheduler processes jobs according to the order of their arrivals. If enough processors are available for running the job at the head of the queue, the scheduler allocates the processors and starts the job. Otherwise, the job should wait for some current running jobs to terminate and free additional processors. FCFS many lead to low utilization of nodes because of idle processors fragmentation. Backfilling [15] is then introduced to make utilization of the idle processors fragmentation. It allows suitable subsequent jobs to use idle nodes when the job at the head of the queue waits for extra resources. Backfilling has been widely used in the industry and many enhanced algorithms [26,27] are proposed to improve the number of jobs to be backfilled. Gang scheduling [16] is another strategy to improve the utilization of nodes, it allows resource sharing among multiple parallel jobs by diving the computing capacity of nodes into time slices. However, it is essentially a resource sharing algorithm which can hardly offer strict bandwidth guarantees. Both backfilling and gang scheduling are designed for clusters which try to improve the utilization of CPU resources and do not consider the allocation of bandwidth resources for jobs.

3. System Model and Problem Definition

3.1. Data Center Network Model

The substrate DCN (Data Center Network) considered in our paper is a multi-rooted hierarchical topology of Fat-Tree [28]. From the bottom up, fat-tree based DCN is divided into three layers: edge layer, aggregation layer and core layer. Given a construction parameter k, the whole configurations of fat-tree are fixed. Every $k/2$ edge switches and $k/2$ aggregate switches together form a pod, and totally k pods are constructed in the network. For each edge switch, the links going up are connected to all the aggregation switches in the same pod and the links going down connect to $k/2$ servers. The links going up of each aggregate switch are connected to $k/2$ core switches. The distinctive feature of a fat-tree is that for any switch, the number of links going down to its siblings is equal to the number of links going up to its parent in the upper level.

We model the DCN as a weighted undirected graph $G(V, E, C(E))$, where V denotes the set of nodes in the network topology, E denotes the set of links between nodes and $C(E) = \{c_e | e \in E\}$ denotes the residual capacities on the links. Let M and W be the set of servers and switches in the data center, respectively, where $M \cup W = V$. The servers are assumed to be homogenous, and each server has L resource slots. Here, the resource slot represents an abstract measurement of CPU, memory, and storage to accommodate a single VM. A corresponding set $A(M) = \{a_s | s \in M\}$ denotes the number of residual resource slots on the servers in M.

3.2. Virtual Cluster Model

In this paper, we deal with resource allocation and scheduling for parallel jobs in the multi-tenant cloud data center. Different with the situation in local clusters and private cloud, the underlying resources in multi-tenant cloud are shared among many different users with different kinds of applications. Resource contentions, especially for network bandwidth, will cause unpredictable execution time of parallel jobs. On the other hand, the execution time estimation of submitted job can be obtained as the bandwidth are guaranteed and is critical for the scheduler to make optimal scheduling decision. Accordingly, a request model is needed for users to specify, including the number of parallel running VMs, bandwidth demand that leads to an expected execution time and the time duration for occupying the resources. Inspired by Ballani et al. [9], Dalvandi et al. [14], a time-aware virtual cluster (TVC) is designed as the virtual cluster request model.

Let $J = \{j_i | i = 1, \dots, R\}$ be the set of submitted parallel jobs over a period of time, where j_i denotes the ith job submitted to the cloud and R denotes the number of jobs in J. For a certain job, it can be represented as $j_i = \{at_i, N_i, B_i, es_i\}$, where at_i denotes the arrive time, N_i denotes the number of VMs requested for running parallel processes, B_i denotes the requested bandwidth for guaranteeing the network performance, and es_i denotes the estimated job execution time under guaranteed bandwidth B_i.

When scheduling a job j_i at time τ_i^{st}, a requested TVC model at τ_i^{st} for job j_i can be represented as $TVC_i = \{N_i, B_i, \tau_i^{s}, \tau_i^{ee}\}$, where N_i denotes the number of VMs, B_i denotes the subscribed bandwidth on virtual links between VMs and the virtual switch, τ_i^{st} denotes the allocating time of virtual cluster for starting j_i and $\tau_i^{ee} = \tau_i^{st} + es_i$ denotes the estimated release time of virtual cluster when j_i finishes running. Different from Reference [14], the TVC model in our paper is designed based on homogenous hose model [29] which can be used without prior knowledge about traffic matrix between the parallel processes. When a TVC is accepted by the cloud, N_i VMs are deployed onto the idle slots of the servers in the cloud data center, and the residual bandwidth capabilities of links along paths routing to the corresponding servers are reduced.

3.3. Problem Definition

In this paper, we design a management and scheduling framework for parallel jobs in the cloud data center, as shown in Figure 1. It consists of three layers: user layer, scheduling layer and resource layer. In the user layer, multiple tenants dynamically submit their jobs to the service provider through web portal and specify jobs' resource demands with TVC model. The submitted job requests are all queued into the job queue at the order of arrival in the scheduling layer. Parallel job scheduler is responsible for deciding which job in the queue should be dispatched according to the characteristics of jobs and the residual resource information from resource monitor. The TVC embedding component is responsible to allocate the VMs of TVC request to the cloud data center while ensuring that the bandwidth demands on the links are satisfied. The resource layer consists of fat-tree based data center and a resource monitor which dynamically tracks the status of residual resources including CPU and bandwidth on the substrate links.

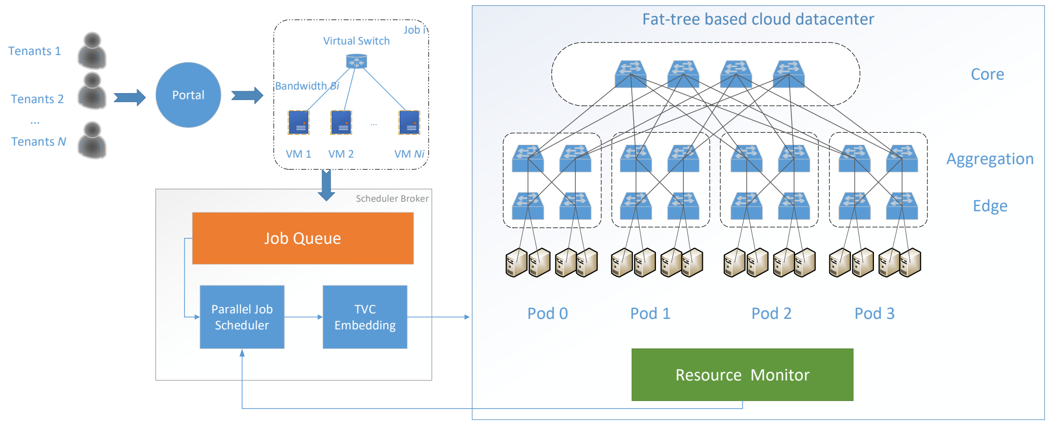


Figure 1. Management and scheduling framework.

The service provider can gain more revenue and bring good user experience by improving the throughput of jobs while guaranteeing the QoS. Accordingly, the objective of the scheduling broker is to minimize the average response time (the duration between arrival and being scheduled) and average slowdown (ratio between execution time of job in the cloud and that in a dedicated cluster) of jobs, while ensuring that the bandwidth and CPU demands are satisfied.

Considering a substrate DCN $G(V, E, C(E))$ and a submitted jobs set $J = \{j_i | i = 1, \ldots, R\}$ where j_i is represented with a TVC model $TVC_i = \{N_i, B_i, \tau_i^{st}, \tau_i^{ee}\}$, the objective of the scheduling broker can be formulated as follows:

$$\min \quad rpt = \frac{1}{R}\sum_{i=1}^{R}(\tau_i^{st} - at_i) \tag{1}$$

$$\min \quad bsd = \frac{1}{R}\sum_{i=1}^{R}\frac{(\tau_i^{st} + \tau_i^{ee} - at_i)}{max(\Gamma, \tau_i^{ee})} \tag{2}$$

where rpt represents the average response time of jobs, and bsd denotes the average bounded slow down which is an optimized version of slow down that mitigates the effect of short jobs by introducing a minimal execution time element Γ [27].

3.4. Constrains

Different with traditional scheduler of clusters, bandwidth resource constraints should be considered when scheduling parallel jobs in the cloud. Given a scheduled job j_i and its TVC request $TVC_i = \{N_i, B_i, \tau_i^{st}, \tau_i^{ee}\}$, a feasible solution of the TVC embedding problem includes VMs placement and traffic routing assignment.

The VMs placement is an assignment of all N_i VMs to the servers without considering the bandwidth constraints. Let $\Pi^{N_i} = \{m_{s,i} | s = 1, \ldots, |M|\}$ denote a VM's placement of TVC request TVC_i, where $m_{s,i}$ denotes the number of VMs allocated to the server s. A feasible VM placement for the TVC request should satisfy the following constraints.

1. The number of allocated VMs should equal the requested number of TVC:

$$\sum_{s=1}^{|M|} m_{s,i} = N_i \tag{3}$$

2. The allocated number of VMs $m_{s,i}$ to server s should not exceed its current number of residual resource slots a_s:

$$m_{s,i} \leq a_s \tag{4}$$

Given a VM placement Π^{N_i}, the traffic route assignment problem is to find feasible placement of virtual switch vs_i of TVC to a substrate switch, where the bandwidth requirement can be accommodated on the substrate links along the path from the virtual switch to each VM. In our paper, we only consider the maximum bandwidth guarantee between servers similar to Reference [9], instead of calculating the ingress and egress traffic flows routing. Because once the bandwidth requirement B_i is given by users, ingress and egress of the traffic flows between parallel processes will not exceed the maximum rate of B_i.

Let $\Omega(\Pi^{N_i}) = \{s | m_{s,i} > 0, s = 1, \ldots, |M|\}$ denote the set of servers that have hosted at least one VM under the VM placement Π^{N_i}, and let $\Pi^{vc_i} = \{x_{w,i} | w = 1, \ldots, |W|\}$ denote the placement of the virtual switch. Placement variable $x_{w,i} = 1$ if the vs_i is assigned on switch w; and $x_{w,i} = 0$ otherwise. Then, a valid placement of virtual switch should satisfy the following constraints.

1. The virtual switch should be assigned to exactly one substrate switch at the same time:

$$\sum_{w=1}^{|W|} x_{w,i} = 1 \tag{5}$$

2. Let $p_{ws,i} = \{e_{l,i} | e_{l,i} \in W\}$ denote the routing path from the switch w where virtual switch locates to server s. According to the Oktopus system [9], the total bandwidth allocated to $p_{ws,i}$ should not exceed the residual bandwidth capability $c_{e_{l,i}}$ of link $e_{l,i}$ along the path $p_{ws,i}$:

$$\sum_{w=1}^{|W|} x_{w,i} \min(m_{s,i}, N_i - m_{s,i}) B_i \leq c_{e_{l,i}} \tag{6}$$

4. Proposed Algorithms

The management and scheduling of parallel jobs in the cloud can be treated as a variant of job scheduling problem that integrates with TVC embedding. Accordingly, it is also a NP-complete problem as it can be obtained by transferring from the parallel job scheduling problem which have been proven to be NP-complete [30]. To minimize the performance metric of rpt and bsd, from the perspective of batch scheduling, the basic idea is to maximize the utilization of idle fragmentations of bandwidth through backfilling algorithm. From the perspective of TVC embedding, the basic heuristic strategy is to maximize the residual bandwidth capabilities that can be used for the following waiting jobs after embedding current TVC. In this paper, we deal with these two issues separately. An adaptive bandwidth-aware algorithm denoted as AdaBa is designed to deal with the TVC embedding problem. A bandwidth-guaranteed migration and backfilling algorithm denoted as BgMBF is devised to maximum the utilization of bandwidth in the cloud.

4.1. Adaptive Bandwidth-Aware (AdaBa) TVC Embedding Algorithm

In this paper, AdaBa is designed to solve the TVC embedding problem when scheduling parallel jobs in the cloud. Considering that the communication of VMs in the same server do not occupy the bandwidth resources on the links, the bandwidth occupation of a virtual cluster can be hidden when its corresponding VMs are all allocated to the same server. From the observation, a basic intuition is that the more virtual cluster request with larger bandwidth demand are hidden, the more bandwidth resources on the links can be reserved for accepting following arrived jobs. Accordingly, based on the intuition, AdaBa tries to embed each virtual cluster request to servers adaptively according to the requested bandwidth demand, ensuring that virtual cluster with largest bandwidth demand is embedded beginning from servers with maximum number of idle slots, and virtual cluster with smallest bandwidth demand is embedded beginning from servers with minimum number of idle slots. In this way, the bandwidth occupation of virtual cluster with larger bandwidth demand has higher probability to be hidden, which offers more opportunity for jobs following to be accepted. To realize the process above, AdaBa assigns each server a weight, and, when a new requested TVC arrives, the serves are sorted in weight descending order which ensures that the server with larger weight have the higher priority to accommodate the VMs of requested TVC. Then, it turns into a weight function designing problem, which ensures that the server with more idle slots has larger weight when the requested TVC with largest bandwidth demand arrives, and, on the contrary, the server with fewer idle slots has larger weight when the requested TVC with smallest bandwidth demand arrives.

To design a proper weight function, two important issues need to be considered. The first one is how to evaluate the relative size of requested bandwidth demand (need to know what is the largest bandwidth demand that can be arrived) and the second one is how to determine the weight of servers when requested bandwidth demands lay between the smallest one and the largest one to conform to the basic idea of AdaBa illustrated above.

To improve the acceptance rate of dynamic arrived jobs, the bandwidth demands that can be arrived in the future should also be considered when embedding current requested TVC. Accordingly, the relative size of requested bandwidth demand in this paper is actually defined as the ratio between the current requested bandwidth demand and the maximum bandwidth demand that can be arrived in the near future. However, the accurate value of the maximum bandwidth demand in the future cannot be obtained in advance. Instead, in AdaBa, we use the value of maximum bandwidth request that has ever arrived, to predict future's maximum bandwidth demand. The value is updated dynamically upon new request is arrived. Accordingly, in AdaBa, we design a weight function as shown in Equation (7), to determine the weight of servers for each requested TVC with different bandwidth demand.

$$weight_s = -0.5{a_s}^2 + L * \frac{B}{B_{max}} * a_s \tag{7}$$

where $weight_s$ denotes the weight of server s, a_s denotes the number of residual slots of the server, B denotes current requested bandwidth demand, L is a constant that represents total number of resource slots in a server and B_{max} represents the maximum bandwidth request that has ever arrived.

When a new requested TVC arrives, the value of L, B_{max} and B are specified, and the value of $weight_s$ is actually a single-variable quadratic polynomial of a_s. The extreme point, which is obtained at $L * \frac{B}{B_{max}}$, means that the servers whose number of residual slots equals $L * \frac{B}{B_{max}}$ are the most suitable locations for current requested TVC. When the bandwidth demand of current TVC is the smallest one, the extreme point of $L * \frac{B}{B_{max}}$ trends to be zero, and the weight of servers increase as its corresponding number of residual slots decrease. On the contrary, when the bandwidth demand of current TVC is the largest one, the extreme point of $L * \frac{B}{B_{max}}$ trends to be L, and the weight of servers increase as its corresponding number of residual slots increase. When the bandwidth demand of current TVC is not the smallest nor the largest, the server whose number of residual slots is closer to $L * \frac{B}{B_{max}} \in (0, L]$ has higher weight, which ensures that the jobs with relative high bandwidth demand (compared with the maximum bandwidth request that has ever arrived) can be allocated to the server with more residual slots.

Figure 2 gives example curves of weight functions under different bandwidth requests, where $B_{max} = 800$ Mbps and $L = 8$. It can be seen that, when the requested bandwidth demand is small enough (where $B = 10$ Mbps and $B = 110$ Mbps), the server with one idle slot achieve the largest weight. As B increases (B varies from 210 Mbps to 710 Mbps in the figure), the number of residual slots of server that achieves the maximum weight (which means the x-axis of extreme point) also increase adaptively. It ensures that, when a requested TVC with relative small bandwidth demand is scheduled, it has higher probability to be allocated to the server with fewer residual slots to fill the fragmentation, and, when a job with relative larger bandwidth demand is scheduled, it has higher probability to be allocated to the server with more residual slots.

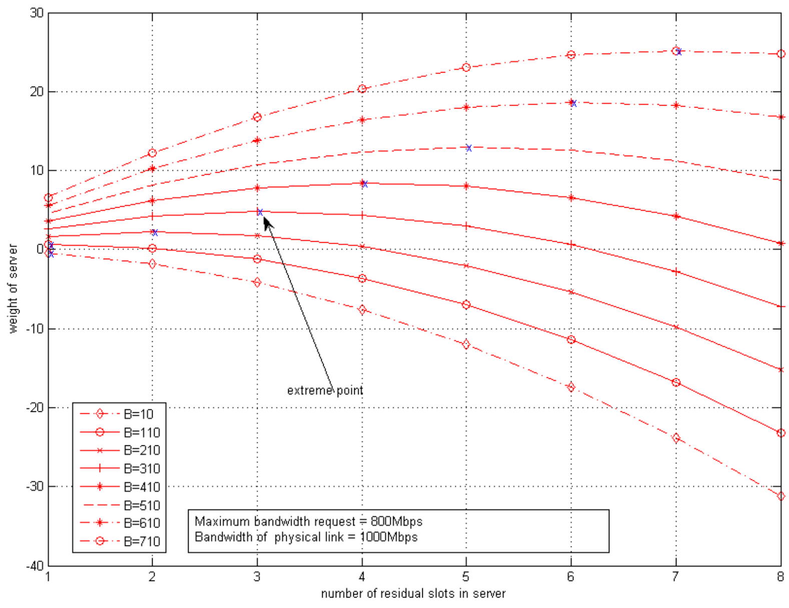
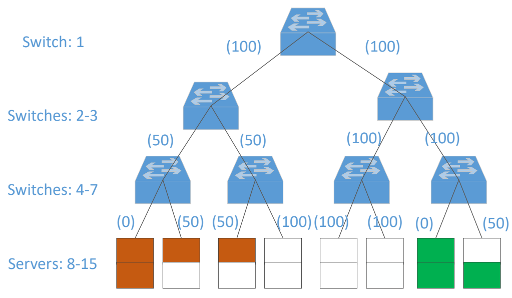


Figure 2. Weight functions of servers under different bandwidth requests.

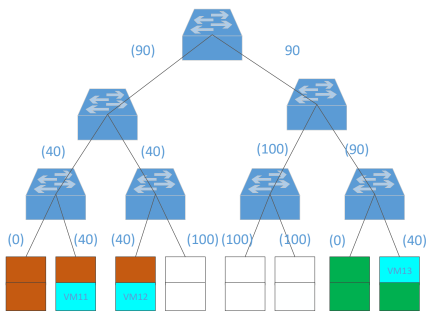
Considering the process of choosing the most suitable location for each requested TVC, AdaBa seems to be similar to best fit strategy. Best fit strategy [31] tries to put the requested job from the current fullest server (with least number of residual slots) in which it fits to avoid so-called "put fine timber to petty use". However, actually, they are only common in the idea of adjusting the priorities of servers dynamically for each requested TVC, and the difference is distinct. In AdaBa, the priorities of servers are adaptively determined according to both the relative size of requested bandwidth demand and the number of residual slots in servers. However, in best fit strategy, the servers are always sorted in increasing order of current number of idle slots without considering the requested bandwidth demands. In other words, they are different in the setting of weight function. In AdaBa, the weight function varies with the relative size of requested bandwidth demand. In best fit strategy, the weight function is seen as invariant and is inversely proportional to current number of residual slots.

On the other hand, different with the greedy method which tries to minimize the bandwidth occupancy on physical links for each job scheduled, AdaBa adjusts the priorities of severs according to a ratio that indicates the relative size of current requested bandwidth demand to realize different embedding strategy according to the bandwidth demand.

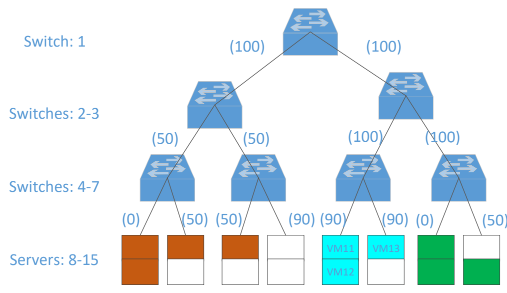
Figure 3 shows a comparison case of different strategies for embedding a TVC request j_i (3, 10 Mbps, 100 s, 120 s). The initial state of the DCN before scheduling j_i is shown in Figure 3a. The two jobs labeled in red and green color are still running in the cloud. The idle slots for accommodating the VMs are shown as white blocks and residual bandwidth for each link is recorded in the bracket beside the link. When scheduling a job j_i with different algorithms, three VMs are allocated to the idle slots which are labeled in blue color and the residual bandwidth on the links are updated, as shown in Figure 3b–d.



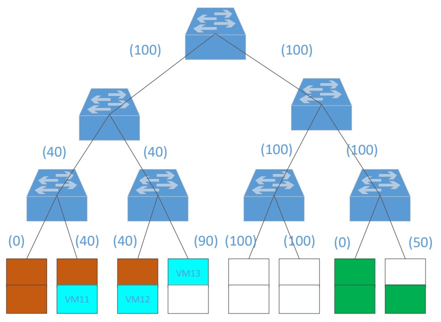
(**a**) state of DCN before scheduling



(**b**) placement result under AdaBa and best fit



(**c**) placement result under greedy strategy



(**d**) placement result under first fit strategy

Figure 3. An comparison of different strategies for embedding a TVC request (3, 10 Mbps, 100 s, 120 s).

It can be seen that, as the requested bandwidth demand is relatively small, AdaBa firstly tries to arrange the VMs of requested TVC to the servers with the least idle slots, and three idle severs are left for accommodating possible arriving jobs with large bandwidth demands (as shown in Figure 3b). For best fit strategy, it sorts the servers in increasing order of current number of idle slots, and arranges the VMs to the sorted servers in sequence. The result is then occasionally the same as AdaBa under this case (as shown in Figure 3b) . For greedy strategy, it obtains the minimum value of maximum link occupation but only one idle sever is reserved (as shown in Figure 3c) . First fit strategy arranges the VMs to the servers in sequence and two idle severs are reserved (as shown in Figure 3d) . Considering three identical TVC requests (2, 60 Mbps, 110 s, 150 s) are arriving at time 110, among these embedding results above under these strategies, only AdaBa and best fit can accept the whole request immediately on their arrivals which obtains the minimum response time.

Algorithm 1 lists the details of AdaBa. The algorithm consists of two parts: VMs placement and virtual switch placement. Given the state of substrate DCN $G(M, W, E, C(E))$, AdaBa will generate the placement plan for VMs and virtual switch for each requested TVC. Firstly, it updates B_{max} to ensure that B_{max} is the maximum bandwidth demand that has ever arrived (see Line 1). The weight of each server can then be calculated according to Equation (7) (see Lines 2–4). After that, AdaBa searches for the possible location of virtual switch placement based on locality-aware strategy [9], which tries to pack the VMs of requested TVC in the smallest sub-tree to minimize the bandwidth reserved at higher levels of the topology. To achieve that, AdaBa traverses the physical switches starting from low level of edge layer (level = 1) to high level of core layer (level = 3), and determines if the requested TVC can be embedded (see Lines 5–29).

Algorithm 1: The adaptive bandwidth-aware TVC embedding algorithm.

```
   Input: $G(M, W, E, C(E)), TVC_i = \{N_i, B_i, \tau_i^s, \tau_i^{ee}\}, B_{max}$
   Output: VMs placement $\Pi^{N_i}$ and virtual switch placement $\Pi^{vs_i}$,$B_{max}$
1  $B_{max} = max(B_{max}, B_i)$;
2  for each $s \in M$ do
3      calculate $weight_s$ by Equation (7);
4  end
5  set the $level = 1$ which is at the edge layer;
6  while $level <= 3$ do
7      get the subset of switches $w_{level}$ at the level from $W$;
8      for each $sw \in w_l$ do
9          get the subset of servers $M_{sw}$ in the subtree of $sw$ from $M$ ;
10         sort $M_{sw}$ in weight descending order ;
11         $leftVm = N_i$, to initialize remaining number of VMs to be allocated;
12         for each $s \in M_{sw}$ do
13             for $m_{s,i} = 1; m_{s,i} < min(a_s, leftVm); m_{s,i} = m_{s,i} + 1$ do
14                 if $m_{s,i}$ satisfy the bandwidth guaranteed constrain by Equation (6) then
15                     $assigned = m_{s,i}$, to record the maximum number of VMs that can be assigned
                       in the server ;
16                 end
17             end
18             $leftVm = N_i - assigned$, to update remaining number of VMs to be allocated ;
19             add $m_{s,i}$ to $\Pi^{N_i}$ ;
20         end
21         if $leftVm = 0$ then
22             $x_{sw,i} = 1$;
23             add $x_{sw,i}$ to set $\Pi^{vs_i}$;
24             return success;
25         else
26             clear set $\Pi^{N_i}$ ;
27         end
28     end
29     $level = level + 1$, to move to higher level of the tree-based network;
30 end
31 return failure;
```

Once the virtual switch is located at current selected physical switch, the servers in the subtree of selected switch are used for allocating the VMs tentatively (see Lines 8–20). If all of the VMs can be accommodated, the selected switch is chosen to be where the virtual switch locates on, and the algorithm ends with success together with the VMs placement Π^{N_i} obtained from the tentative VMs placement process (see Lines 21–24). Otherwise, clear Π^{N_i} and continue to try next switch. The algorithm ends with failure when there is no successful placement of VMs for all the switches.

In the VMs placement process, the servers in the subtree are sorted in weight descending order which ensures that the server with larger weight has higher priority to accommodate the requested VMs. Then, the algorithm tries to allocate the VMs to the sorted server in sequence. When allocating VMs to a server, it tries to maximize the number of VMs that can be allocated, while ensuring that the bandwidth constraints on the links along the path from selected switch to the server are not violated (see Lines 13–17). The calculated number $m_{s,i}$ that records the number of VMs assigned in server s is then added to the VMs placement set Π^{N_i}.

4.2. Bandwidth-Guaranteed Migration and Backfilling (BgMBF) Scheduling Algorithm

In this section, we propose a bandwidth-guaranteed migration and backfilling scheduling algorithm (denoted as BgMBF) to schedule parallel jobs running in cloud efficiently. The algorithm is designed based on FCFS principle to ensure the fairness among the jobs. However, in FCFS principle, when the top job is waiting for the release of extra resources (including CPU and bandwidth), the computational capabilities and bandwidth resources in idle fragmentations are wasted. Backfilling algorithm is a popular way to improve the utilization of fragmentation resources which tries to fill the idle resources with jobs in the waiting queue. However, traditional backfilling algorithm and its variant never guarantee the bandwidth demands of jobs. Accordingly, we design BgMBF by integrating existing AMBF algorithm [17] with our proposed AdaBa to maximize the utilization of resources in the cloud while guaranteeing the bandwidth demands of scheduled jobs. The basic idea of BgMBF is to relax the strict FCFS constraint, where suitable jobs following the top job are allowed to be backfilled running in the idle fragmentations when the top job is waiting for releasing extra resources. Additional, a preempting mechanism is designed to ensure that the top job should not be delayed by backfilled jobs, and avoid possible job starvation. The preempting process is triggered when the total resources including both the idle resources and resources that occupied by backfilled jobs, are enough to embed the TVC request of the top job. The backfilled jobs that are being preempted should then be suspended and pushed back into the job waiting queue to be rescheduled again. Migration supported by the cloud is used to suspend and recover the VMs that belong to the preempted jobs across the servers.

Algorithm 2 lists the details of the scheduling process. The algorithm is activated when a new job arrives or a job finishes execution. Given current state of substrate DCN (denoted as G) and current placement of jobs running in the DCN (denoted as P), BgMBF determines when and where the jobs in the job waiting queue (denoted as Q) can be scheduled running in the cloud. Firstly, BgMBF schedules the jobs in Q starting from the top one and determines if it can be accepted running in the cloud with current idle resources through AdaBa. If it is successfully accepted, the scheduler will remove it from the queue and dispatch it running in the cloud according to the placement results from AdaBa (see Lines 4–5). It should be noted that the current index of job i should be reduced by 1 to get a correct index for the next job when current job is removed from Q (see Line 10). Otherwise, if current job is the top one, BgMBF will further check whether the preempting process can be conducted (see Line 12). To realize that, BgMBF sets up two temporary states, $G_{withoutBF}$ and $P_{withoutBF}$, from G and P by removing the placement of current backfilled jobs running in the cloud, and determines if the top job can be accepted under $G_{withoutBF}$ and $P_{withoutBF}$ through AdaBa (see Lines 13–14). If the job can be successfully accepted, the preempting process is triggered that all the VMs of jobs in $Q_{backfill}$ are suspended and the jobs are moved back to Q, and the state of substrate DCN and current placement of jobs are updated accordingly(see Lines 16–18). The top job is then removed from the queue and scheduled to the cloud according to the allocation placement from AdaBa.

Algorithm 2: Bandwidth-guaranteed migration and backfilling scheduling algorithm.

```
Input: $G(M, W, E, C(E))$, $P$ , $Q$
Output: the updated placements of jobs running in DCN $P'$
1  $i = 1$, to get the top job $j_i$ from $Q$;
2  while $TVC_i \neq null$ do
3      embedding $TVC_i$ by AdaBa (see algorithm 1) under current $G$ and $P$;
4      if success then
5          remove $j_i$ from $Q$ and dispatch it running in the cloud according to the placement
           results including $\Pi^{N_i}$ and $\Pi^{vs_i}$ ;
6          update $M$ and $G$ accordingly ;
7          if $i \neq 1$ then
8              record the placement state of $j_i$ to $Q_{backfill}$;
9          end
10         $i = i - 1$, to update the index as $j_i$ is removed from $Q$;
11     else
12         if $i = 1$ then
13             obtain $P_{withoutBF}$ and $G_{withoutBF}$ from $P$ and $G$ respectively by removing the
               placement of jobs in $Q_{backfill}$ ;
14             embedding $TVC_i$ by AdaBa under current $G_{withoutBF}$ and $P_{withoutBF}$ ;
15             if success then
16                 suspend all the VMs of jobs in $Q_{backfill}$ and move the jobs back to $Q$;
17                 $P = P_{withoutBF}$ ;
18                 $G = G_{withoutBF}$ ;
19                 remove $j_i$ from $Q$ and dispatch it running in the cloud according to the
                   placement results including $\Pi^{N_i}$ and $\Pi^{vs_i}$ ;
20                 update $P$ and $G$ accordingly ;
21                 $i = i - 1$, to update the index as $j_i$ is removed from $Q$ ;
22             end
23         end
24     end
25     $i = i + 1$, to get next job in $Q$;
26 end
```

No matter if current job is embedded successfully or not, the scheduling process will not stop to wait for the release of extra resources, but go on scheduling the jobs in the queue tentatively in arrival order (see Line 25). Any jobs that can be successfully scheduled running in the cloud before the top job are normally removed from the queue and dispatched running in the cloud, but, additionally, they should be recorded in $Q_{backfill}$ for possible preempting process use (see Lines 7–9).

The algorithm above is a general algorithm as it can be used without the estimated execution time of jobs. It chooses the backfilled jobs in the waiting queue in arrival order. However, it may be inefficient as the backfilled jobs are expected to be preempted frequently when the execution duration of chosen jobs are relatively large.

In this paper, when the bandwidth demand of job is guaranteed, its execution time then can be predicted or a minimum execution time can be obtained [9]. Accordingly, an enhanced algorithm can be obtained by running the SDF (short duration first) heuristic strategy that the job in the waiting queue with minimum estimated execution time has the highest priority to be backfilled. Through the method, the jobs backfilled in the fragmentation have higher probability to be finished before being preempted, which leads to higher average response time and smaller migration cost.

5. Performance Evaluation

5.1. Simulation Settings

In this section, we evaluate the performance of our proposed algorithms using trace-driven simulation. A discrete event-based simulator is designed with Matlab to implement the management and scheduling framework for parallel jobs proposed in this paper.

In the simulation, the substrate DCN architecture is designed based on fat-tree, but our proposed algorithms can be also extended for other DCN architectures. We set the constructing number k [28] of fat-tree to be 6, which implies that there are $k = 6$ pods; each pod has $k/2 = 3$ aggregation switches and $k/2 = 3$ edge switches; and each aggregation switch is connected to $k/2 = 3$ core switches. For each edge switch, there connects $k/2 = 3$ servers and the servers are homogenous that each has eight slots for accommodating VMs. The line rate for all the substrate links are set to be 1 Gbps. Accordingly, the substrate DCN in our simulation has 432 slots to accommodate the node request of parallel jobs.

5.2. Workload

In our simulation, we use following two real parallel workloads logs from production systems:

1. KTH SP2 workload logs: This log contains eleven months of accounting records from the 100-node IBM SP2 at the Swedish Royal Institute of Technology (KTH) in Stockholm. We select the first 1000 jobs to generate parallel workload in our simulation and the average job runtime is 72.4 min.
2. NASA iPSC: This log contains three months of records for the 128-node iPSC/860 located in Numerical Aerodynamic Simulation (NAS) Systems Division at NASA. We select the first 1000 jobs to generate parallel workload in our simulation and the average job runtime is 10.4 min.

As the workloads above do not contain the bandwidth demand information, we assign the bandwidth demand for each job according to the following process:

1. Calculate the maximum bandwidth demand that can be scheduled in the cloud by equation $Max_B = NetworkBandwidth * \frac{NumberofServers}{NumberofNodesRequested}$.
2. Calculate the minimum bandwidth demand by equation $Min_B = \frac{Max_B}{10}$ and the average bandwidth $Avg_B = \frac{Min_B + Max_B}{2}$.
3. Generate the bandwidth demand through a normal distribution $B = max(min(Min_B, normrnd(Avg_B, 0.2 * Avg_B)), Max_B)$ and make sure the result do not exceed the maximum and minimum constrain.

Different workload characteristics of bandwidth demand have different influences on the experimental results. However, we can still use the process above to generate the bandwidth demand for the workloads without affecting in a negative way the experimental results. The reasons can be summarized as follows:

1. The bandwidth demands of jobs are generated by normal distribution which is a popular way to simulate the bandwidth demands in the current literature [9,10]. It is a reasonable way to simulate the probability distribution of bandwidth demands in reality.
2. In addition to the probability distribution characteristic, another important characteristic of the bandwidth demands in the generated workloads is the average bandwidth demand. We have conducted sufficient experiments to evaluate the impact of different average bandwidth demand on the performance for different algorithms. The result shows that the relative performance index of different algorithms keep invariant generally as the average bandwidth demands of workload vary (see Figure 4). Accordingly, the experimental results under the workloads with reasonable probability distribution of bandwidth demands can be representative to evaluate performance of different algorithms.

The detailed information of above two workloads in our simulation are shown in Table 1. Through multiplying the interval time between two arrival jobs by a ratio, the system load of the cloud data center can be easily changed to evaluate scheduling performance under different situations.

Table 1. Workloads in the simulation.

	Average Job Runtime	Number of Jobs	Average Bandwidth	Maximum Node Request
KTH-SP2	72.4 m	1000	251 Mbps	100
NASA-iPSC [32]	10.4 m	1000	247 Mbps	128

5.3. Results and Discussion

5.3.1. Performance of TVC Embedding

In this section, we evaluate the performance of our proposed TVC embedding algorithm of AdaBa. To avoid the influence of scheduling strategy on the embedding performance, we conduct the experiment without job queue, which implies that, if a job cannot be allocated upon its arrival, it is rejected. Accordingly, the accept rate is used as the metric to evaluate the TVC embedding performance. In the experiment, we compare our proposed AdaBa with three other popular competitors, including Oktupus algorithm [9], greedy algorithm [10] and best fit strategy. Oktupus algorithm is designed based on locality-aware strategy and greedy algorithm tries to assign the VMs to the server in the DCN that leads to the minimum value of maximum link occupation. Best fit strategy always sorts the servers in current number of residual slots and tries to accommodate the requested TVC in sequence from servers with minimum number of residual slots. Best fit strategy is designed to compare the performance of different settings of weight function.

Considering the TVC embedding performance should be highly correlated with the bandwidth demands, we firstly evaluate the impact of different average bandwidth demands of workloads on the performance. As the average bandwidth demand of workloads in Table 1 is fixed, we generate a serial of random parallel workloads whose average bandwidth demands are varied from 50 to 700 Mbps, and their average node requests and the system loads are all fixed to be 8 and 0.5, respectively. The comparison results can be found in Figure 4. From the figure, we have following observations: (1) Our proposed AdaBa always performs better than all the other algorithms for workloads under different average bandwidth demands; (2) As the average bandwidth demand goes higher, the superiority of AdaBa gets more significant. Up to 15 percent of improvement on accept rate is obtained when the average bandwidth demand is 700 Mbps; (3) The accept rate decreases as the average bandwidth demand increases. However, the rate of descent of our proposed AdaBa gets slower significantly, which guarantees higher performance when the bandwidth demand goes higher compared with other algorithm; (4) Best fit also performs better than Oktupus and greedy strategy in most cases, and the decreasing tendency of best fit strategy is similar with AdaBa. However, AdaBa still outperforms best fit strategy significantly.

From the observations, our proposed bandwidth-aware strategy of AdaBa performs the best in different average bandwidth demands, as the weight of severs are adaptively changed according to the ratio of current arrived bandwidth demand and the maximum one that has ever been scheduled. As the average bandwidth demand increases, the bottleneck of resource allocation turns from CPU to bandwidth, and the advantage of AdaBa gets more significant. The superiority should owe to the proposed bandwidth hidden mechanism of AdaBa. When the average bandwidth demand goes higher, AdaBa reserves as many idle slots as possible in one server to accommodate possible arriving jobs with large bandwidth demand, which reduces the resource contention of bandwidth on the link significantly. As large bandwidth occupation can be hidden, AdaBa is less insensitive to the increase of average bandwidth demand. Accordingly, the rate of descent of AdaBa will get slower when the average bandwidth demand increases. Best fit strategy is also a method that changes the weight of severs adaptively. It tries to deploy the VMs of requested TVC to the servers with minimum number of residual slots. In other words, in best fit, the server with fewer residual slots has larger weight. Accordingly, it performs the same with AdaBa when the bandwidth demand of requested TVC is relatively small. On the other hand, it can also hide bandwidth occupation occasionally as it always reserves the servers with more idle slots. This is why that best fit strategy performs better than Oktupus and greedy strategy, especially when the average bandwidth demand is relatively high. However, the value of hidden bandwidth occupation is uncontrollable and unpredictable, as the best fit strategy determines the weight servers without considering current requested bandwidth demand. Differently, in order to control the value of hidden bandwidth occupation, AdaBa changes the weight function of server dynamically according to the relative size of current requested bandwidth demand (see Figure 2). The extreme point of the weight function under current requested TVC then represents the optimal location of VMs to hide as much bandwidth occupation of requested TVC with large bandwidth demand as possible. Accordingly, compared with first fit, AdaBa still obtains considerble advantage in accept rate.

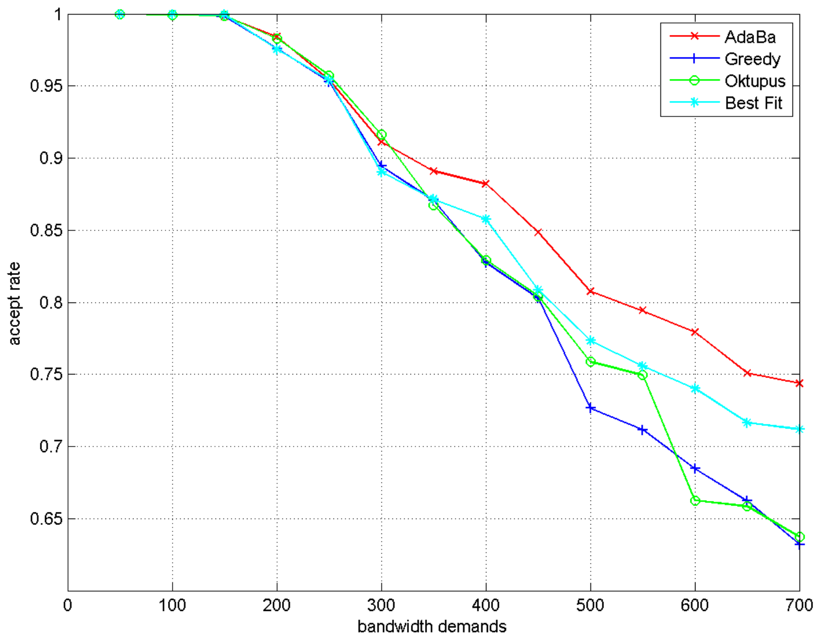
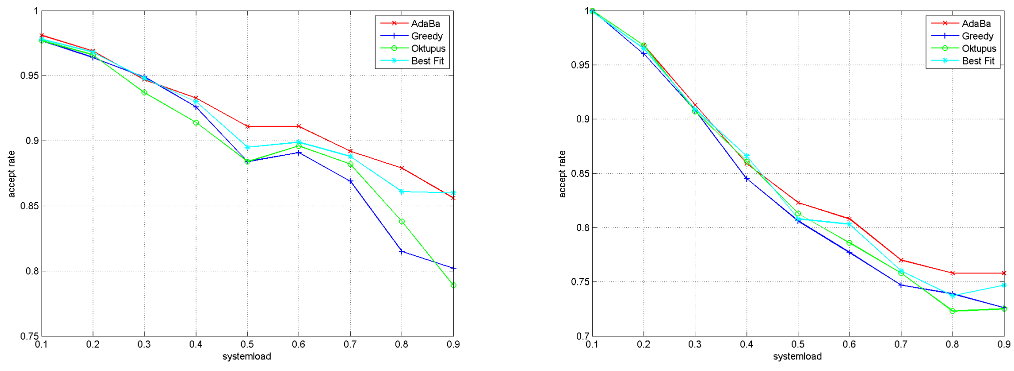


Figure 4. Comparison of accept rate for workloads with different average bandwidth demands.

Furthermore, we investigate the performance of AbaBa for the realistic workloads in Table 1, and the results are shown in Figure 5. These experiments are conducted under different system loads while the average bandwidth demands and other characteristics of jobs are fixed. In the figure, we can find that our proposed AdaBa algorithm also performs better than other competitors under most cases of system loads for each workload. On the other hand ,the accept rate improvement of AdaBa is getting more significant as the system load goes higher. That is because AdaBa considers not only current TVC embedding request but also maximizing the residual bandwidth capabilities that can be used for the following waiting jobs, and, accordingly, the resource contention will mitigate especially when the system load is relatively high. However, as the average bandwidth demand of these workloads are all around 250 Mbps which can not fully exert the advantage of AdaBa, as shown in Figure 4, the maximum acceptance rate improvements of AbaBa are not that significant.

From the discussions above, the relative size of requested bandwidth demand, which is obtained by introducing the prediction of maximum bandwidth demand in the near future, is the key factor for AdaBa to achieve the performance. The result verifies the intuition that the resource allocation decision for dynamic arriving workloads can be more efficient by synthesizing the resource demands in the future. However, the performance also relies on the prediction accuracy of future resource demand.

However, static bandwidth reservation method in this paper on the links may be still inefficient when the volumes of traffic flows among the parallel processes of jobs are dynamically changed with a large range. Accordingly, if the volume of traffic flows among the parallel processes of jobs can be predicted or estimated at a high level of confidence, a more fine-grained dynamic bandwidth allocation mechanism, which dynamically consolidates complementary bandwidth demands on the links, is expected to be effective in improving the utilization of bandwidth resource on the links without violating the QoS of tenants. However, there are still many difficult problems (such as traffic flows prediction and fault-tolerant problem) to be solved in the future to realize the method.



(**a**) accept rate with KTH SP2 (**b**) accept rate with NASA iPSC

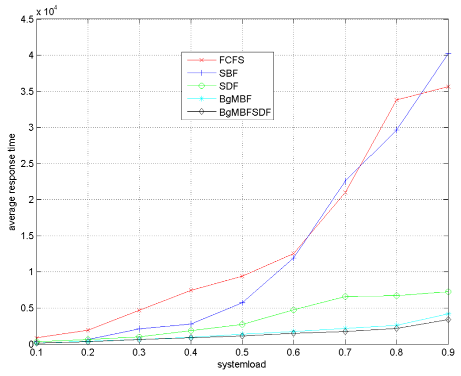
Figure 5. Comparison of the acceptance rate under different workloads.

5.3.2. Scheduling Performance

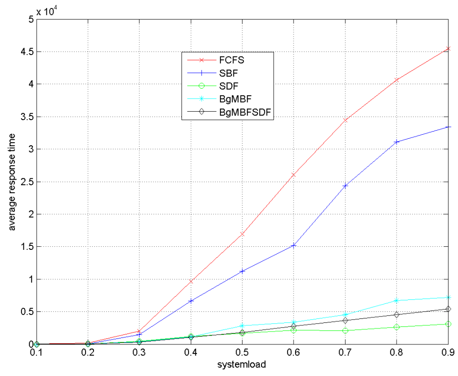
In this section, we further evaluate the performance of our proposed bandwidth-guaranteed migration and backfilling scheduling algorithm (denoted as BgMBF) and its enhanced version of BgMBFSDF that selects the backfilling jobs with short duration first (SDF) strategy. The bandwidth request of jobs are guaranteed by our proposed AdaBa because of its advantage on improving the accept rate for dynamic arriving parallel jobs as shown in previous section. Three other competitor are used for comparison: FCFS, SBF (short bandwidth demand first) and SDF (short duration first). The scheduling objective of rpt (Equation (1)) and bsd (Equation (2)) are used as the performance metrics. The performance comparisons for KTH-SP2 and NASA-iPSC are shown in Figures 6 and 7. From the figures, we have the following observations:

1. Our proposed BgMBF significantly outperforms FCFS and SBF on average response time and average bounded slow down for each workload. It shows the great advantage of backfilling that tires to maximize the utilization of idle resources caused by the FCFS strategy. It also shows that the small bandwidth demand first strategy cannot improve the scheduling performance compared with FCFS, and it slows down the scheduling time of jobs with large bandwidth demand. On the other hand, the performance differences among BgMBFSDF, SDF and BgMBF are relatively small. However, BgMBFSDF always performs better than BgMBF. It demonstrates that the heuristic information of estimated execution time is important for improving the scheduling performance. As the jobs with small duration are expected to be finished before the next arriving jobs, it leads to high responsiveness performance, and shortens the idle time of fragmentation resources which improves the utilization of resource. However, SDF is an unfair strategy which will cause starvation of jobs with long execution time. However, when the estimated execution time is known, it can be used for improving the performance of BgMBF by conducting BgMBFSDF strategy while guaranteeing the fairness among jobs.

2. SDF outperforms BgMBFSDF and BgMBF on average response time for NASA-iPSC, while obtains the worst performance among them for KTH-SP2. Compared with BgMBF, the performance improvement on *rpt* of BgMBFSDF is also more significant for NASA-iPSC than that for KTH-SP2. The differences above are caused by the different characteristics of these two workloads. In Table 1, it can be seen that the average job runtime for KTH-SP2 and NASA-iPSC is 72.4 min and 10.4 min, respectively. Accordingly, the advantage of SDF is more significant for NASA-iPSC, as more jobs with short duration can be finished before next arriving job comes. For BgMBFSDF strategy, more jobs with smaller duration means more jobs can be finished in advanced through backfilling without migration.

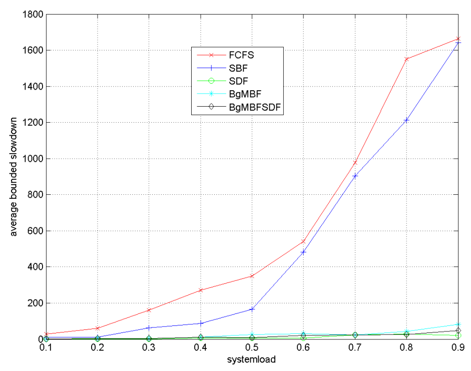


(**a**) *rpt* with KTH SP2

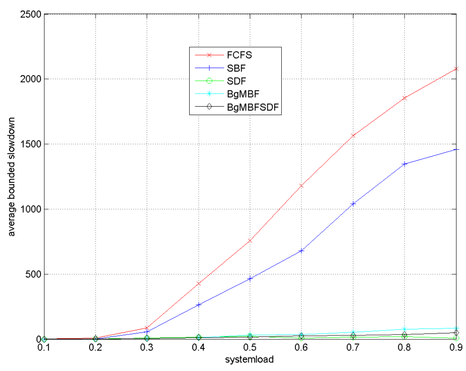


(**b**) *rpt* with NASA iPSC

Figure 6. Comparison of the average response time with different workloads.



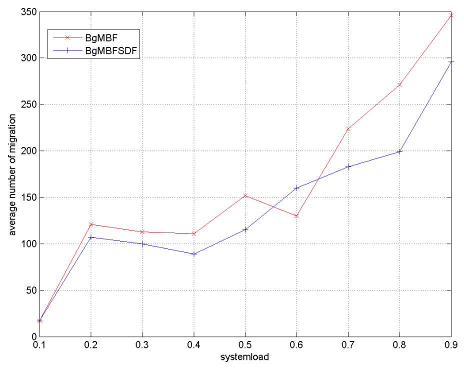
(**a**) *bsd* with KTH SP2



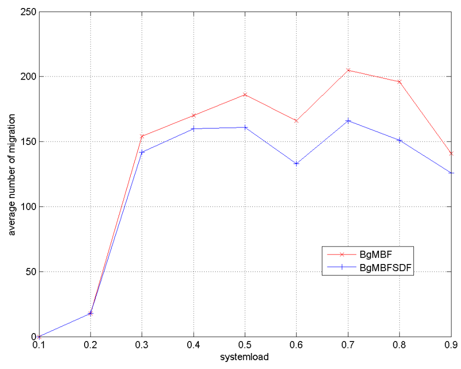
(**b**) *bsd* with NASA iPSC

Figure 7. Comparison of the average bounded slowdown with different workloads.

Furthermore, we record the total number of migrations for BgMBF and BgMBFSDF, respectively, to evaluate effect of migration and make comparison between BgMBF and BgMBFSDF. The comparison results are shown in Figure 8. The maximum number of migrations are 350 and 200 for KTH-SP2 and NASA-iPSC, respectively. Considering the migration cost in the cloud is round 20 s [33], the time cost for migration in our proposed BgMBF is $o(10^3)$ which is far smaller than the average response time of $o(10^4)$. Accordingly, it is worthy and acceptable to improve the scheduling performance by migration and backfilling method. On the other hand, the number of migrations in BgMBFSDF is less than that in BgMBF in most cases except when the system load is 0.6 for workload KTH-SP2. As BgMBFSDF chooses the jobs with small duration to backfill into the idle fragmentation resources, the jobs are expected to be finished without migration before being preempted with higher probability compared with BgMBF. Since the average job runtime of KTH-SP2 is large than of NASA-iPSC, the number of jobs that need to be migrated for KTH-SP2 is less than that for NASA-iPSC at the same system load.



(**a**) number of migrations with KTH SP2



(**b**) number of migrations with NASA iPSC

Figure 8. Comparison of the migration number with different workloads.

6. Conclusions

In this paper, we focus on the management and scheduling issues of parallel jobs in the cloud data center. Guaranteeing the bandwidth demands of scheduled jobs running in the cloud leads to predictable job execution performance and batch scheduling performance, and is also the major motivation of this paper. In this paper, we have proposed a management and scheduling framework for parallel jobs in the cloud. To realize the virtual cluster embedding component, which is responsible to allocate requested VMs and reserve requested bandwidth resource, we design an adaptive bandwidth aware (AdaBa) virtual cluster embedding algorithm. By adjusting the priorities of servers to accommodate the VMs of TVC adaptively according to the relative size of requested bandwidth demand, AdaBa hides the bandwidth occupation of jobs with large bandwidth demand on the substrate links. AdaBa always outperforms the competitors for workloads with different average bandwidth demands and system loads. As the average bandwidth demand of submitted jobs increases, the advantage of AdaBa becomes more significant. Accordingly, compared with other virtual cluster embedding algorithms, the ability of adaptive bandwidth-aware of AdaBa shows great advantage for parallel jobs, as it always evolves a large range of bandwidth demands. On the other hand, we design a bandwidth-guaranteed migration and backfilling (BgMBF) algorithm to schedule parallel jobs running in the cloud. AdaBa is integrated into BgMBF to allocate requested VMs and reserve requested bandwidth on the substrate link. BgMBF improves the average response time by scheduling the suitable jobs in the waiting queue with the idle fragmentation resources in advance. When the estimated execution time is available, BgMBFSDF, which selects the backfilled jobs with SDF strategy, is proposed to improve BgMBF. It increases the number of backfilled jobs that finish before being preempted and reduces the migration cost. From the simulation results, BgMBFSDF always outperforms BgMBF on average response time and average bounded slow down.

In our future work, we plan to extend AdaBa to solve the TVC embedding problem with more general situation, such as heterogeneous bandwidth demand of TVC and bandwidth demand uncertainty. The dynamic bandwidth allocation mechanism for parallel jobs is also another direction of future work.

Author Contributions: Zhen Li, Bin Chen and Xiaocheng Liu contributed to developing ideas about the framework with algorithms; Dandan Ning helped conduct the experiments; Qihang Wei and Yiping Wang helped analyze and discuss the experiment data; Xiaogang Qiu supervised the paperwork, review etc. All authors were involved in preparing the manuscript.

Acknowledgments: This study is supported by National Key Research & Development (R&D) Plan under Grant No. 2017YFC0803300; the National Natural Science Foundation of China under Grant Nos. 71673292, 71673294, 61503402, and 61673388; National Social Science Foundation of China under Grant No. 17CGL047; and Guangdong Key Laboratory for Big Data Analysis and Simulation of Public Opinion. The workload log from the KTH SP2 was graciously provided by Lars Malinowsky (lama@pdc.kth.se), who also helped with background information and interpretation.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

G	substrate DCN (data center network)
V	the set of nodes in network topology
E	the set of links between nodes
c_e	the residual capacity on link e
M	the set of servers, where $M \subset V$
W	the set of switches, where $W \subset V$
a_s	the number of residual resource slots on server s, where $s \in M$
L	resource slots in a idle server
J	the set of submitted jobs
at_i	the arrive time of the i_{th} job
N_i	the number of VMs requested for the i_{the} job
B_i	the bandwidth requested for the i_{the} job
es_i	estimated job execution time of the i_{th} job
τ_i^{st}	start time for the i_{th} job
τ_i^{ee}	end time for the i_{th} job
rpt	average response time of J
bsd	the bounded slow down of J
Π^{N_i}	a VMs placement of TVC_i
Π^{vs_i}	a virtual switch placement of TVC_i
$m_{s,i}$	the number of VMs allocated to server s
$x_{w,i}$	the assignment variable if vs_i is located to switch w
$p_{ws,i}$	the routing path between switch w and server s

References

1. Pellerin, D.; Ballantyne, D.B.A. *An Introduction to High Performance Computing on AWS: Scalable, Cost-Effective Solutions for Engineering, Business, and Science*; Amazon Web Service: Seattle, WA, USA, 2015.
2. Gupta, A.; Sarood, O.; Kale, L.V.; Milojicic, D. Improving HPC Application Performance in Cloud through Dynamic Load Balancing. In Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Delft, The Netherlands, 13–16 May 2013; pp. 402–409.
3. Walker, E. Benchmarking amazon EC2 for high-performance scientific computing. *Usenix* **2008**, 18–23.
4. Gupta, A.; Milojicic, D. Evaluation of HPC Applications on Cloud. In Proceedings of the Sixth Open Cirrus Summit, Atlanta, GA, USA, 12–13 October 2011; pp. 22–26.
5. Somasundaram, T.S.; Govindarajan, K. CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud. *Future Gener. Comput. Syst.* **2014**, *34*, 47–65. [CrossRef]
6. Zhao, Z.; Martin, P.; Wang, J.; Taal, A.; Jones, A.; Taylor, I.; Stankovski, V.; Vega, I.G.; Suciu, G.; Ulisses, A.; et al. Developing and Operating Time Critical Applications in Clouds: The State of the Art and the SWITCH Approach. *Procedia Comput. Sci.* **2015**, *68*, 17–28. [CrossRef]
7. Zhao, Z.; Martin, P.; Jones, A.; Taylor, I.; Stankovski, V.; Salado, G.F.; Suciu, G.; Ulisses, A.; de Laat, C. Developing, Provisioning and Controlling Time Critical Applications in Cloud. In *Advances in Service-Oriented and Cloud Computing*; Mann, Z.Á., Stolz, V., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 169–174.
8. Guo, C.; Lu, G.; Wang, H.J.; Kong, C.; Sun, P.; Wu, W.; Zhang, Y. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In Proceedings of the 6th International Conference, Philadelphia, PA, USA, 30 November–3 December 2010; p. 15.

9. Ballani, H.; Costa, P.; Karagiannis, T.; Rowstron, A. Towards predictable datacenter networks. *Acm Sigcomm Comput. Commun. Rev.* **2011**, *41*, 242–253. [CrossRef]
10. Yu, L.; Shen, H.; Cai, Z.; Liu, L.; Pu, C. Towards Bandwidth Guarantee for Virtual Clusters under Demand Uncertainty in Multi-tenant Clouds. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 450–465. [CrossRef]
11. Biran, O.; Corradi, A.; Fanelli, M.; Foschini, L.; Nus, A.; Raz, D.; Silvera, E. A Stable Network-Aware VM Placement for Cloud Systems. In Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Ottawa, ON, Canada, 13–16 May 2012; pp. 498–506.
12. Yan, F.; Lee, T.T.; Hu, W. Congestion-Aware Embedding of Heterogeneous Bandwidth Virtual Data Centers With Hose Model Abstraction. *IEEE/ACM Trans. Netw.* **2017**, *PP*, 1–14. [CrossRef]
13. Li, X.; Wu, J.; Tang, S.; Lu, S. Let's stay together: Towards traffic aware virtual machine placement in data centers. In Proceedings of the IEEE INFOCOM, Toronto, ON, Canada, 27 April–2 May 2014; pp. 1842–1850.
14. Dalvandi, A.; Gurusamy, M.; Chua, K.C. Time-Aware VMFlow Placement, Routing, and Migration for Power Efficiency in Data Centers. *IEEE Trans. Netw. Serv. Manag.* **2015**, *12*, 349–362. [CrossRef]
15. Lifka, D.A. The ANL/IBM SP Scheduling System. In Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, CA, USA, 25 April 1995; pp. 295–303.
16. Feitelson, D.G.; Jette, M.A. Improved Utilization and Responsiveness with Gang Scheduling. In Proceedings of the Job Scheduling Strategies for Parallel Processing, Geneva, Switzerland, 5 April 1997; pp. 238–261.
17. Liu, X.; Wang, C.; Zhou, B.B.; Chen, J.; Yang, T.; Zomaya, A.Y. Priority-Based Consolidation of Parallel Workloads in the Cloud. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *24*, 1874–1883. [CrossRef]
18. Popa, L.; Yalagandula, P.; Banerjee, S.; Mogul, J.C.; Turner, Y.; Santos, J.R. ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing. *Comput. Commun. Rev.* **2013**, *43*, 351–362. [CrossRef]
19. Rodrigues, H.; Santos, J.R.; Turner, Y.; Soares, P.; Guedes, D. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In Proceedings of the Conference on I/o Virtualization, Berkeley, CA, USA, 14 June 2011; p. 6.
20. Shieh, A.; Saha, B. Sharing the data center network. In Proceedings of the Usenix Conference on Networked Systems Design and Implementation, Boston, MA, USA, 30 March–1 April 2011; pp. 309–322.
21. Chen, L.; Feng, Y.; Li, B.; Li, B. Towards performance-centric fairness in datacenter networks. *Int. J. Hybrid Intell. Syst.* **2013**, *10*, 23–32.
22. Li, D.; Chen, C.; Guan, J.; Zhang, Y.; Zhu, J.; Yu, R. DCloud: Deadline-Aware Resource Allocation for Cloud Computing Jobs. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 2248–2260. [CrossRef]
23. Yu, R.; Xue, G.; Zhang, X.; Li, D. Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers. In Proceedings of the INFOCOM 2017—IEEE Conference on Computer Communications, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
24. Feitelson, D.G.; Rudolph, L.; Schwiegelshohn, U. Parallel Job Scheduling—A Status Report. In Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, New York, NY, USA, 13 June 2004; pp. 1–16.
25. Schwiegelshohn, U.; Yahyapour, R. Analysis of First-Come-First-Serve Parallel Job Scheduling. In Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, 25–27 January 1998; pp. 629–638.
26. Gómez-Martín, C.; Vega-Rodríguez, M.A.; González-Sánchez, J. Fattened backfilling: An improved strategy for job scheduling in parallel systems. *J. Parallel Distrib. Comput.* **2016**, *97*, 69–77. [CrossRef]
27. Mu'Alem, A.W.; Feitelson, D.G. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *Parallel Distrib. Syst. IEEE Trans.* **2001**, *12*, 529–543. [CrossRef]
28. Al-Fares, M.; Loukissas, A.; Vahdat, A. A scalable, commodity data center network architecture. *Acm Sigcomm Comput. Commun. Rev.* **2008**, *38*, 63–74. [CrossRef]
29. Duffield, N.G.; Goyal, P.; Mishra, P.; Ramakrishnan, K.K.; Merwe, J.E.V.D. Resource management with hoses: Point-to-cloud services for virtual private networks. *IEEE/ACM Trans. Netw.* **2002**, *10*, 679–692. [CrossRef]
30. Du, J.; Leung, Y.T. Complexity of Scheduling Parallel Task Systems. *Siam J. Discret. Math.* **1989**, *2*, 473–487. [CrossRef]
31. Kenyon, C. Best-fit bin-packing with random order. In Proceedings of the Acm-Siam Symposium on Discrete Algorithms, Atlanta, GA, USA, 28–30 January 1996; pp. 359–364.

32. Feitelson, D.G.; Nitzberg, B. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. *Lect. Notes Comput. Sci.* **1970**, 337–360.
33. Ye, K.; Jiang, X.; Ma, R.; Yan, F. VC-Migration: Live Migration of Virtual Clusters in the Cloud. In Proceedings of the ACM/IEEE International Conference on Grid Computing, Beijing, China, 20–23 September 2012; pp. 209–218.