Goodbye to Fixed Bandwidth Reservation: Job Scheduling with Elastic Bandwidth Reservation in Clouds

Haiying Shen*, Lei Yu†, Liuhua Chen‡, Zhuozhao Li*
*Department of Computer Science, University of Virginia
{hs6ms, zl5uq}@virginia.edu
†College of Computing, Georgia Institute of Technology
leiyu@gatech.edu
‡Department of Electrical and Computer Engineering, Clemson University
liuhuac@clemson.edu

***Abstract*—The shared nature of cloud network infrastructures causes unpredictable network performance, which may degrade the performance of these applications. Recently, several works propose to explicitly reserve the network bandwidth in the cloud with virtual network abstraction models, which pre-specify the network bandwidth between virtual machines (VMs) for a tenant job. However, the pre-specification fails to exploit the elastic feature of the bandwidth resource (i.e., more reserved bandwidth within no-elongation threshold bandwidth leads to shorter job execution time and vice versa) in job scheduling. It is difficult for ordinary tenants (without specialized network knowledge) to estimate the exact needed bandwidth. In this paper, we propose a new cloud job scheduler, in which each tenant only needs to specify job deadline and each job's reserved bandwidth is elastically determined by leveraging the elastic feature to maximize the total job rewards, which represent the worth of successful completion by deadlines. Finally, the scheduler tries to reduce the execution time of each job. It also jointly considers the computational capacity of VMs and reserved VM bandwidth in job scheduling. Using trace-driven and real cluster experiments, we show the efficiency and effectiveness of our job scheduler in comparison with other scheduling strategies.**

I. Introduction

The shared and multi-tenant nature of cloud network leads to significantly varying data transmission latency due to bandwidth competitions among network flows, which further causes unpredictable application performance [1–4]. Without bandwidth guarantee, the job completion time could be two orders of magnitude longer [5]. To address the performance predictability, several works [6–8] proposed to augment cloud computing with virtual network service models and explicitly enable bandwidth reservation in datacenter networks.

Previous work [8] shows that a lower VM bandwidth for a job may lead to longer networking time and hence longer job execution time. Despite the fact that the job execution time and the VM bandwidth of network abstractions have such dependency, there exists little work that exploits the relationship between them for high performance in bandwidth reservation. Also, the bandwidth pre-specification is not suitable for ordinary tenants who may not have specialized network knowledge because it is difficult for them to estimate the exact needed bandwidth. Rather than the VM bandwidth, they most likely provide a job description with the required job deadline and possibly a reward value representing the worth of completing the job by its deadline [9, 10]. The reward can be determined based on different objectives such as maximizing the number of completed jobs, the number of service-level agreement (SLA) conformances, the social welfare of customers or the profit of the cloud provider. Then, based on this provided information, it is the cloud provider's responsibility to decide appropriate VM bandwidth in the virtual network abstraction.

In this paper, we propose a new cloud service model that novelly leverages the elastic feature of the bandwidth resource (i.e., the dependency between job execution time and the VM bandwidth of a virtual network abstraction) to maximize the total rewards of jobs (referred to as *system reward*) and also reduce job execution time. In this model, tenants only need to specify the job deadline and reward value if it is determined by the tenants instead of the cloud provider. Based on this information, for each job, the model determines the VM bandwidth in the virtual network abstraction, the PM to allocate each VM of the job, and the job start time.

In the framework, we leverage job profiling to model the relationship between the job execution time and VM bandwidth, i.e., the job execution time corresponding to each level of provisioned VM bandwidth for each job. With the input of such job profiles and job descriptions, the job scheduler decides a scheduling of cloud resources for running jobs. To determine the optimal schedule that maximizes the total system reward, we formulate the problem. Due to its hardness, we propose an approximate algorithm for scheduling. The algorithm first uses the smallest VM bandwidth level of each job that meets its deadline in job allocation. Then, it increases the VM bandwidth level (i.e., reduce execution time) of each job starting from the longest job gradually if its VM allocation is still valid until job execution time cannot be reduced anymore. Finally, it uses the highest level of VM bandwidth for each job starting from the job with the most VMs if its VM allocation is still valid in order to quickly release more resources to allocate to other jobs.

The rest of the paper is organized as follows. Section II presents our cloud service model and the system framework.

In Section III, we formulate the job scheduling problem. Section IV presents our scheduling algorithm in detail. In Section V, we evaluate our algorithm in comparison with a number of typical scheduling strategies. Section VI presents the related work and Section VII concludes this paper.

II. System Mode and Framework

In this section, we describe the service model and our system framework that integrates the bandwidth reservation into the job scheduling.

A. Service Model

1) Job Model: In our service model, each tenant rents a set of homogenous VMs to run his job. Each job or service request from the tenant, denoted by j, is characterized by a tuple $\{N_j, d_j, v_j\}$, where N_j is the number of VMs the tenant requires, d_j is the desired deadline of job j, and v_j is the job's reward value that represents the worth of a successful completion of the job before its deadline d_j. As the tenants usually specify the number of VMs when they buy VMs, it is reasonable to assume that the number of VMs is fixed in the model. The jobs with a fixed number of VMs are also known as *rigid jobs* in the parallel scheduling research [11]. The reward value can be determined by the tenant or the cloud provider. The goal of our model is to determine a schedule of jobs under cloud capacity constraints to optimize a reward value based objective function. In this paper, we choose the sum of reward values of the jobs that are fully executed before their deadlines as the objective function. The jobs that are completed after their deadlines yield zero value.

2) Virtual Network Abstraction: To characterize the bandwidth requirement of a job, we use a simple virtual cluster abstraction proposed in Oktopus [7], as shown in Figure 1. The virtual cluster abstraction, represented by $< N, B >$, describes a virtual topology comprising of N machines connected by bidirectional links of capacity (i.e., VM bandwidth) B to a virtual switch. That is, each VM can send and receive at rate B. For each job j in a set of jobs to allocate, given the input of $\{N_j, d_j, v_j\}$, the cloud scheduler generates the appropriate virtual cluster abstraction $< N_j, B_j >$ in offline scheduling, which is used in resource allocation for the job. The elastic feature means that a job has different execution times with different reserved bandwidth resources. B_j is determined so that the system reward is maximized and also the execution time is reduced as much as possible.

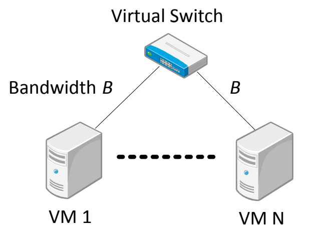


Fig. 1: Virtual cluster abstraction $< N, B >$.

B. System Framework

As shown in Figure 2, our system framework for realizing the above service model consists of three components.

Job profiler. A novelty of this paper is that we consider the elastic feature of the bandwidth resource to determine the reserved bandwidth in order to optimize a reward value based objective function for the jobs that complete before their deadlines. We will explain the details in Section II-C.

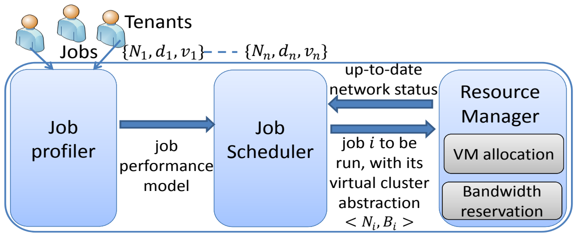


Fig. 2: System framework.

Job Scheduler. Given a batch of jobs and their profiles, the job scheduler is responsible for determining the optimal schedule of jobs. The job scheduler must have up-to-date information of available resources in the datacenter. A job cannot be scheduled if the datacenter does not have enough resources to support the corresponding virtual cluster abstraction. Thus, the job scheduler needs to obtain the up-to-date network status information from the resource manager. The job scheduler informs the resource manager of the determined schedule.

Resource Manager. Resource manager maintains and reports to the job scheduler the up-to-date information of the datacenter resource, including the datacenter network topology, the empty VM slots in each physical machine, and the residual bandwidth on each physical link, calculated from tallying the resource allocations of currently running jobs. It also performs the VM allocation and bandwidth reservation given the scheduling decisions from the job scheduler. The mechanism to enforce the resource reservation in the datacenter for the virtual network abstractions has been well developed in previous works [7, 8], and thus we assume that the resource manager uses an existing mechanism to fulfill its task.

C. Job Profiler

As indicated previously, instead of pre-specifying the bandwidth demand, we only require tenants to specify the job deadline. Then, we determine the amount of VM bandwidth for each job to determine its completion time in order to maximize a reward value based objective function for jobs completed by their deadlines. Therefore, it is necessary to learn the relationship between the job execution time and the VM bandwidth for each job. For this purpose, we use the job profiler for profiling each application offline. Given the type of VMs, the number of VMs determines the computational capacity of the virtual cluster. Because the number of VMs is pre-specified by the tenant, the computational capacity for running a job is pre-determined and hence the execution time of a specific job largely depends on the inter-VM flow completion time, which is decided by the VM bandwidth.

Several previous works [7, 12] show that the measurements from profiling runs are good predictors of the application traffic pattern in production. The application profiling has been well exploited to optimize the resource allocation [12–14]. Recent work shows that the profiles of jobs can be obtained with high accuracy [13, 15] and some open-source tools such as Starfish [16] have been developed to generate such profiles.

Therefore, we assume that the job profiler can have reasonably accurate estimate of the application's communication pattern online. Like [13], we only schedule those jobs whose profiles can be obtained with high accuracy.

Profiling Methodology Suppose a set of VM bandwidth options $\{B_1, B_2, \ldots, B_m\}$ in increasing order for jobs' virtual cluster abstractions. Previous work [8] has found that, for each application, until the VM bandwidth is reduced to a certain threshold (referred to as *no-elongation threshold bandwidth*), there is virtually no impact on the application execution time because of the limitation of application data generation rate. Once the VM bandwidth drops below the threshold, the execution time is elongated monotonically. It suggests that the VM bandwidth for a job should always be the same or lower than the non-elongation threshold bandwidth, since higher bandwidth wastes networking resource without speeding up the job. Therefore, any VM bandwidths larger than the no-elongation threshold should not be selected by the job scheduler for a job. To this end, the job profiler needs to identify the no-elongation threshold bandwidth for a job, with the results obtained from the profiling runs.

Output For each job, say j, the output of job profiler for it is a sequence of tuples $\{(B_1^j, \tau_1^j), (B_2^j, \tau_2^j), \ldots, (B_k^j, \tau_k^j)\}$ where each tuple consists of an available VM bandwidth and the corresponding job execution time, and the bandwidth $B_1^j < B_2^j < \ldots < B_k^j$. The largest bandwidth B_k^j is the no-elongation threshold bandwidth and $\tau_1^j > \tau_2^j > \ldots > \tau_k^j$.

III. Job Scheduling Problem

In this section, we focus on our job scheduling problem. We first present the system model, and then formulate the problem as an optimization problem [17] and show its hardness.

A. System Model

The datacenter network is abstracted by a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where V is the set of physical machines and switches, and E is the set of physical links that connect machines and switches. In this paper, we focus on tree-like topologies such as multi-rooted tree [18, 19], which are typical topologies of today's datacenters. We assume that the datacenter has M homogeneous physical machines. Each physical machine offers S VM slots. Figure 3 shows an example of the tree topology. Time is divided into equal-length time slots. The length of a time slot is the minimum time granularity. For simplicity, the length of a time slot is set to unit 1.

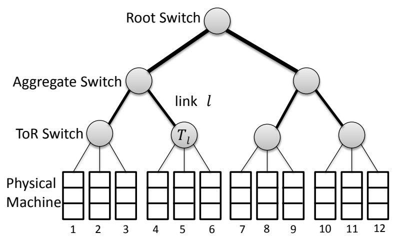


Fig. 3: Tree topology of the datacenter. Each PM has 3 VM slots.

Consider that there are n jobs. Each job j ($j = 1, \ldots, n$) is associated with a tuple (N_j, d_j, v_j) and its performance profile $\{(B_1^j, \tau_1^j), (B_2^j, \tau_2^j), \ldots, (B_{k_j}^j, \tau_{k_j}^j)\}$. As we indicated previously, N_j is the number of VMs required by job j, d_j is j's deadline, and v_j is its value. The performance profile describes the job execution time of j (τ_i^j) under k_j different VM bandwidth settings $B_1^j, \ldots, B_{k_j}^j$ in increasing order.

B. Problem Formulation

Our job scheduling problem is to determine an optimal schedule of jobs under the cloud capacity constraints. It specifies the VM bandwidth, the time to run, and VM placement in datacenter for each job, with the goal to maximize the total value of jobs that are completed before their deadlines. The problem can be formulated as a 0-1 Integer Program.

Decision Variables: For job j, to characterize the decision of the time to execute a job and its VM bandwidth, we have the following 0-1 variables:

$$e_{jt} = \begin{cases} 1, & \text{if job } j \text{ starts execution in time } t \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$b_{ji} = \begin{cases} 1, & \text{if job } j \text{ uses VM bandwidth } B_i^j \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Suppose the VMs of job j are numbered from 1 to N_j, and the physical machines of the datacenter are numbered from 1 to M. Then, a VM placement can be represented by the 0-1 variable:

$$p_{mi}^j = \begin{cases} 1, & \text{if job } j \text{ has VM } i \text{ placed on PM } m \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Constraints: Because each job j will be scheduled to start in only one time slot, we have constraint for decision variable e_{jt} in (1)

$$\sum_{t=t_0}^{T} e_{jt} = 1, \quad \forall\ j \quad (4)$$

where t_0 is the time to start executing jobs and T is the upper bound on the duration of the time scheduled for all job. Similarly, a job has only one VM bandwidth from its job profile $\{(B_1^j, \tau_1^j), (B_2^j, \tau_2^j), \ldots, (B_k^j, \tau_k^j)\}$ in its virtual network abstraction

$$\sum_{i=1}^{k_j} b_{ji} = 1, \quad \forall\ j \quad (5)$$

A VM of job j can only be placed on one PM, so

$$\sum_{m=1}^{M} p_{mi}^j = 1, \quad \forall\ j, i \quad (6)$$

A schedule is valid only if it satisfies the cloud capacity constraints. That is, at any time, i) the number of VMs placed on a PM cannot exceed the number of VM slots, and ii) the bandwidth reserved for the jobs on a link cannot exceed its capacity. To formulate the capacity constraints, we first need to derive an indicator to represent whether job j is still running at time t. Given the decision variables (1) and (2), job j's start time and execution time can be represented by $\sum_{t=t_0}^{T} te_{jt}$ and $\sum_{i=1}^{k_j} b_{ji}\tau_i^j$, respectively. If job j is still running at time t, then we must have $t - \sum_{i=1}^{k_j} b_{ji}\tau_i^j < \sum_{t=t_0}^{T} te_{jt} \le t$, that is,

$$\sum_{y=t-\sum_{i=1}^{k_j} b_{ji}\tau_i^j}^{t} e_{jy} = 1$$

Otherwise, the left side of the above equation equals 0. Thus, we use $I_{jt} = \sum_{y=t-\sum_{i=1}^{k_j} b_{ji}\tau_i^j}^{t} e_{jy}$ as the indicator to represent whether job j is still running at time t.

Suppose the job scheduler has the up-to-date network resource information, received from the resource manager. The information includes the number of available VM slots on each PM m and the residual bandwidth of each link l, denoted by S_m and C_l respectively. Then, for each PM m, the number of used slots by all jobs running on PM m must be no more than S_m. Accordingly,

$$\sum_j \sum_{i=1}^{N_j} I_{jt} p_{mi}^j \leq S_m, \quad \forall\ 1 \leq t \leq T, \forall\ m \tag{7}$$

which indicates the constraints of available VM slots on PM m for the schedule.

To address the constraint of link capacity, with a given tree topology of the datacenter, we define a constant c_{lm} for each pair of link l and PM m. If PM m is the leaf node of the subtree T_l connected by l to the upper level, $c_{lm} = 1$; otherwise, $c_{lm} = 0$. Take Figure 3 for instance. c_{l4}, c_{l5}, c_{l6} are "1" since the PMs 4, 5, 6 are in the subtree T_l.

For each link l, the reserved bandwidth for a virtual cluster $< N, B >$ (N is the number of VMs and B is the VM bandwidth) is $\min(N - X_l, X_l) \times B$, where X_l is the number of VMs contained in the subtree T_l connected by the link l. For job j, the number of VMs contained in the subtree T_l, denoted by $X_l j$, can be represented with decision variable (3), that is,

$$X_{lj} = \sum_{i=1}^{N_j} \sum_{m=1}^{M} c_{lm} p_{mi}^j \tag{8}$$

Also, the VM bandwidth of job j is decided by $\sum_{i=1}^{k_j} b_{ji} B_i^j$. Then, we can formulate the constraints of link capacity as

$$\sum_j I_{jt} \min(X_{lj}, N_j - X_{lj}) \sum_{i=1}^{k_j} b_{ji} B_i^j \leq C_l, \quad \forall 1 \leq t \leq T, \forall\ l \tag{9}$$

Objective: We aim to maximize the total reward value of jobs that are completed before their deadlines, named *system reward*. Thus, we first derive the indicator to represent whether a job is completed by its deadline based on the decision variables. Note that job j's start time and execution time equal to $\sum_{t=t_0}^{T} te_{jt}$ and $\sum_{i=1}^{k_j} b_{ji} \tau_i^j$, respectively. If job j is finished by its deadline d_j, the Inequality $\sum_{t=t_0}^{T} te_{jt} \leq d_j - \sum_{i=1}^{k_j} b_{ji} \tau_i^j + 1$ must hold, that is,

$$\sum_{t=t_0}^{d_j - \sum_{i=1}^{k_j} b_{ji} \tau_i^j + 1} e_{jt} = 1. \tag{10}$$

Then, we let $I_j = \sum_{t=t_0}^{d_j - \sum_{i=1}^{k_j} b_{ji} \tau_i^j + 1} e_{jt}$ be the indicator. For a job j which misses its deadline, $I_j = 0$; Otherwise, $I_j = 1$. Consequently, our optimization objective becomes

$$\max \sum_j I_j v_j \tag{11}$$

where v_j is job j's reward value.

IV. Our Scheduling Algorithm

In this section, we propose an efficient heuristic algorithm to address the job scheduling problem in Section III. This offline algorithm assumes that the scheduler has full knowledge of n jobs in a batch before computing. Given the jobs and the available resource information in the datacenter, the algorithm determines the optimal schedule.

The pre-specification may lead to bandwidth capacity overload or underload. To avoid bandwidth competition and fully utilize the bandwidth resource, we first use the smallest bandwidth option in a job's profile what meets its deadline in job scheduling and then increase the bandwidth in the job's profile until its allocation becomes invalid, i.e., either its computation or bandwidth resource demand cannot be satisfied. The second step consists of two sub-steps. First, among the scheduled jobs, from the longest job, it chooses the next higher bandwidth option (i.e., shorter execution time) in its profile if its allocation is still valid. Second, when no job's execution time can be reduced, it tries to use the maximum admissible bandwidth option for each job. The reason that we do not directly apply the second sub-step to minimize the job execution time one by one without the first sub-step is because of the shared nature of the datacenter networks. Increasing the VM bandwidth to the maximum admissible bandwidth for one job may leave no space for the bandwidth increase of other jobs that share the links with this job. But by repeatedly reducing the longest job execution time with only using the next higher bandwidth option, we have a chance to fairly distribute the residual bandwidth of a link to all jobs.

All jobs in a batch may not be able to be scheduled at the starting time t_0 at the same time due to limited resources. Also, jobs will release their allocated resources when they complete during T. Therefore, the scheduling algorithm executes at each $t_i \in T$ to allocate the VM cluster abstraction of unscheduled jobs. After the process of scheduling jobs starting from t_0 is completed, the algorithm conducts scheduling the remaining unscheduled jobs starting from t_1 using the same process. Such process repeats until all jobs are scheduled or T is reached.

To find a valid allocation in the datacenter for a job's virtual cluster abstraction, we take advantage of existing algorithms for the allocation of virtual cluster abstractions [8], rather than reinventing the wheel. In [8], a Dynamic Programming (DP) based VM allocation algorithm is proposed, which aims to find the lowest subtree for virtual cluster allocation in a datacenter with tree topology. By searching the lowest possible subtree that satisfies both slot and bandwidth requirements, the DP algorithm finds the most localized allocation of VMs so as to conserve the bandwidth of the links in the upper levels of the tree and maximize the ability to accommodate the next coming jobs. For a virtual cluster $<N_j, B_{min}^j>$ of job j, the DP algorithm traverses the topology tree starting at the leaves (physical machines at level 0, as shown in Figure 3) and determines if all N_j VMs can fit under the link capacity constraints. We consider this VM allocation algorithm as a procedure call **VMplace**.

Algorithm 1 shows the pseudo-code of our algorithm. Let J_{ready} be a set of n jobs ready to be scheduled in the system at time t_0. For each job $j \in J_{ready}$, we suppose its deadline is regulated to the value relative to the time to start scheduling t_0, denoted by $\hat{d}_j$. Our scheduling algorithm first determines a set of jobs, which can be concurrently allocated and executed at time t_0, denoted by J_{run}. $ResourceStatus$ is the description of the empty VM slots in each physical machine and the residual bandwidth on each physical link.

Algorithm 1 Job scheduling algorithm.

```
Input: $J_{ready}$ – a set of jobs ready to be scheduled, each with $(N_j, \hat{d}_j, v_j)$
    and profile $\{(B_1^j, \tau_1^j), (B_2^j, \tau_2^j), \ldots, (B_{k_j}^j, \tau_{k_j}^j)\}$.
Output: a schedule $\mathcal{S}$ for the jobs.
 1: for each $t_k$ $(0 \le t_k < T)$ do
 2:     /*Allocate each job using the minimum bandwidth option that meets its deadline*/
 3:     $J_{run} \leftarrow \emptyset$;
 4:     $ResourceStatus \leftarrow$ Resource status at $t_k$ from resource manager;
 5:     Sort jobs in $J_{ready}$ in non-increasing order of $\frac{v_j}{\hat{d}_j}$;
 6:     for job $j$ from 1 to $|J_{ready}|$ do
 7:         $B_{min}^j \leftarrow \min\{B_i^j \mid \tau_i^j < \hat{d}_j, 1 \le i \le k_j\}$;
 8:         if $B_{min}^j = \emptyset$ then
 9:             $J_{ready} \leftarrow J_{ready} \setminus j$;
10:         else
11:             $B_j \leftarrow B_{min}^j$; {$B_j$ is the VM bandwidth of job $j$}
12:             $\tau_j \leftarrow \tau_{min}^j$; {$\tau_j$ is the job execution time of job $j$ with $B_j$}
13:             $P_j \leftarrow$ VMplace($< N_j, B_j >, ResourceStatus$);
14:             if $P_j \ne \emptyset$ then
15:                 $J_{run} \leftarrow J_{run} \cup \{j\}$;
16:                 Update $ResourceStatus$ with $j$'s allocation;
17:             end if
18:         end if
19:     end for
20:     /*Reduce the execution time of jobs by increasing bandwidth*/
21:     repeat
22:         Find the longest job $j_L$ in $J_{run}$;
23:         $\tau_{temp} \leftarrow \tau_{j_L}$;
24:         $B_{temp} \leftarrow$ next higher bandwidth in $j_L$'s profile than $B_{j_L}$;
25:         if $B_{temp} \ne \emptyset$ and $P_{j_L}$ is valid given $B_{temp}$ then
26:             $B_{j_L} \leftarrow B_{temp}$;
27:             $\tau_{j_L} \leftarrow$ the job execution time corresponding to $B_{temp}$;
28:         end if
29:     until $\tau_{temp} \le \tau_{j_L}$
30:     /*Maximally reduce the execution time of jobs by using their maximum bandwidth options*/
31:     for each job $j \in J_{run}$ in non-increasing order of num of VMs do
32:         $B_j \leftarrow \max\{B_i^j | B_i^j > B_j$ and $P_j$ is valid given $B_i^j, 1 \le i \le k_j\}$;
33:     end for
34:     $J_{ready} \leftarrow J_{ready} \setminus J_{run}$;
35:     $\mathcal{S} \leftarrow J_{run}$ with $B_j$ and $P_j$ for $\forall j \in J_{run}$;
36: end for
37: Return $\mathcal{S}$;
```

First, for each job $j \in J_{ready}$, the algorithm chooses the smallest bandwidth option from its profile that can satisfy the deadline requirement as j's initial VM bandwidth, denoted by B_{min}^j (Lines 7-12). That is, for the job execution time corresponding to B_{min}^j, denoted by τ_{min}^j, $\tau_{min}^j < \hat{d}_j$ must be satisfied. If any bandwidth options in job j's profile cannot satisfy the deadline requirement, the job is discarded from J_{ready} (Lines 8-9). Then, the algorithm tries to find valid allocations for each job j in J_{ready} with the virtual cluster abstraction $<N_j, B_{min}^j>$ one by one (Lines 11-17). The algorithm here allocates jobs in non-increasing order of their *reward-deadline ratios*, defined as $\frac{v_j}{\hat{d}_j}$. In this way, a job with a higher reward value and a smaller deadline (i.e., greater urgency) can have a higher priority for resource allocation. Obviously, if each job has a reward value "1", the algorithm allocates jobs by EDF essentially.

The scheduling algorithm calls **VMplace** for each job in J_{ready} one by one in non-increasing order of their reward-deadline ratios (Line 13). For every job j that has a valid VM placement for its virtual cluster abstraction $<N_j, B_{min}^j>$, the algorithm puts j into J_{run} and records its VM placement P_j (Line 15). Besides the real resource status of the datacenter at t_0 received from the resource manager, the scheduling algorithm needs to maintain a virtual resource status, which takes into account the VM allocations for jobs in current J_{run}. Each time when a job j is added into J_{run}, the virtual resource status is updated according to its VM placement and virtual cluster abstraction (Line 16). Also, the algorithm needs to pass the up-to-date virtual resource status to **VMplace** to find a valid allocation for next job.

So far, J_{run} includes all jobs scheduled with their smallest bandwidth options that can meet their deadlines. Next, the algorithm optimizes the VM bandwidth selections for jobs in J_{run} with the goal to minimize their execution times. It consists of two sub-steps: i) reducing the longest job execution time among all jobs (Lines 20-29); and ii) minimizing the execution time for each individual job (Lines 30-33). In the first sub-step, the algorithm determines the longest job that has the largest execution time in J_{run}, say j_L, under current VM bandwidth assignment, and tries to increase its VM bandwidth to the next higher bandwidth option in its profile. If the next bandwidth option for j_L is admissible under its VM placement P_{j_L}, i.e, the bandwidth will not exceed the link capacity, then it is used as j_L's new VM bandwidth and j_L's execution time is updated. The algorithm repeats the above procedure after each update, until it cannot further reduce the execution time of the longest job either because the next bandwidth option will exceed the link capacity or because all options run out. In the second sub-step, the algorithm fetches all jobs one by one in non-increasing order of the number of VMs N_j, and uses the maximum admissible bandwidth option for each job's VM bandwidth to minimize the job execution time. In this way, a job with a larger number of VMs can be finished earlier and release its resources sooner. As a result, the remaining jobs in J_{ready} can be scheduled at an earlier time and thus have a better chance to meet their deadlines as well as increase the system reward.

Finally, the algorithm has determined the set of jobs J_{run} that can be concurrently allocated and run at time t_0. The scheduler updates the schedule $\mathcal{S}$ with the set J_{run} along with VM bandwidth B_j and VM placement P_j for each job in it (Lines 34-35). Then, the algorithm schedules the remaining jobs in J_{ready} for time t_1 (Lines 1-36). After the scheduling for each $t_i \in T$ has completed or all jobs have been scheduled, the scheduler informs the resource manager about the schedule $\mathcal{S}$ (Line 37). Accordingly, the resource manager performs the VM allocation and the bandwidth reservation, and starts the jobs accordingly.

V. Performance Evaluation

In this section, we present our trace-driven simulation and real cluster implementation results of our Job Scheduling Algorithm, referred to as JSA for short. We compared JSA with a number of typical methods. One is a typical scheduling strategy, Earliest Deadline First (EDF), which allocates the jobs in non-decreasing order of their deadlines. The second compared method is a virtual cluster allocation algorithm, Oktopus [7], which tries to place all the VMs from a job in a smallest sub-tree. The sub-tree has sufficient empty VM slots and the available bandwidth of the link that connects the sub-tree to the rest of the network satisfies the bandwidth demand of the VMs. The third compared method is Proportional Sharing on Proximate Links (PS-P) in FairCloud [20]. PS-P allocate the link bandwidth to each VM when a link is congested in order to achieve proportional sharing on proximate links while providing minimum guarantee. It assigns the communication between VMs X and Y on a link L based on assigned weight: $W_{X-Y} = W_{Y-X} = \alpha \frac{W_X}{N_X} + \beta \frac{W_Y}{N_Y}$, where N_X (N_Y) is the number of other VMs that X (Y) communicates through link L, and W_X (W_Y) is the weight of VM X (Y). PS-P prioritizes VMs that are close to a given link. More precisely, PS-P uses $\alpha = 1$ and $\beta = 0$ for all links in the tree that are closer to X than Y, and $\alpha = 0$ and $\beta = 1$ for all links closer to Y than X. To extend PS-P for job scheduling, we let it use Oktopus to allocate jobs. Even when a link's bandwidth is used up, as long as there are available slots, PS-P still allocates job VMs and will handle the link congestion as introduced above. We evaluate the performance of these methods in the scenario of scheduling five batches of jobs in a datacenter system.

A. *Experiment Setup*

Workload. We used the Facebook synthesized workload [21] to generate a sequence of jobs running on a 24-machine cluster. We ran 2500 jobs from the 6000 jobs in the Facebook workload consecutively on the cluster and measured the job information. We then used the measured job information to set the parameters in order to conduct a rigorous trace-driven simulation and real implementation experiment. The number of VMs needed by each job j, i.e., N_j, was set to the number of tasks of the job measured from the trace. The reward value of each job v_j was randomly chosen from a uniform distribution within an interval of [100,1000].

For the job performance profiles, we measured the average bandwidth usage $\bar{B}$ over the execution time period (denoted by τ_1^j) of the tasks of the job. We generated the set of bandwidth options for each job from $\bar{B}$ by multiplying a ratio, i.e., {0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0}. For each job j, the execution time τ_1^j under the VM bandwidth $\bar{B}$ was measured from the experiments. For simplicity, the execution time under other bandwidth options was determined by a linear speedup model, $\tau_i^j = (1 - \alpha_j * \frac{B_i^j - \bar{B}}{0.2\bar{B}}) * \tau_1^j$ ($i \neq 1$), where α_j (called speedup ratio) is randomly chosen from (0,0.14) for every job j at default. Larger α_j (i,e., a higher speedup ratio) means a given bandwidth increase leads to more job execution time decrease; a given bandwidth decrease results in more execution time increase.. The deadline $d_j = \beta \tau_1^j$, where β is randomly chosen from [2,7] at default. EDF, Oktopus and PS-P use $\bar{B}$ as predicted bandwidth demand for job scheduling.

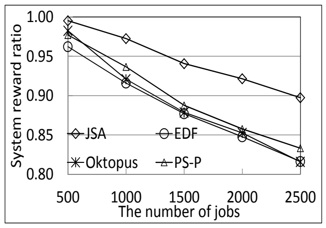
Simulation. We simulate a datacenter of three-level tree topology, as shown in Figure 3. A rack consists of 40 machines; each with 4 VM slots and a 1Gbps link to connect to a Top-of-Rack (ToR) switch. Every 20 ToR switches are connected to a level-2 aggregation switch and 2 aggregation switches are connected to the datacenter core switch. As the setup in [7], there are total 1600 machines at level 0. The link bandwidth between a ToR switch and an aggregation switch is 8Gbps and the link bandwidth between an aggregation switch and the core switch is 32Gbps. It means that the oversubscription of the physical network is 5.

Real cluster implementation. We implemented our JSA job scheduling and evaluate it on a 24-node cluster on CloudLab cluster [22]. We emulated the 24-node cluster in a three-level tree topology with the realistic bandwidth. We divided the 24-node cluster into 4 sub-clusters of 6 nodes each. Each sub-cluster is emulated as a rack and each node has 1 VM slots and connects to a Top-of-Rack (ToR) switch with a 1Gbps link. Every 2 ToR switches are connected to a level-2 aggregation switch and 2 aggregation switches are connected to the datacenter core switch. The link bandwidth between a ToR switch and an aggregation switch is 1.2Gbps and the link bandwidth between an aggregation switch and the core switch is 0.48Gbps. It means that the oversubscription of the physical network is 5. We identified each node by a number. We use the network utility tools *tc* and *iptables* to implement these bandwidth limits. In this experiment, we simply the jobs as data transfer jobs. We generated the jobs based on the job information we collected from the Facebook workload. The transferred data size and the time to transfer of each job are based on the shuffle data size and the execution time of the job. In addition to the 24-node cluster, we used another node to calculate the offline schedule. At the beginning, the batches of jobs were submitted to this node first, which then calculates the schedule and the bandwidth for each job.

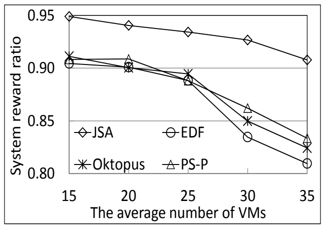
B. *Experimental Results*

We call the ratio of the total reward value of the jobs that complete before their deadlines to the total reward value of all jobs *system reward ratio*. The results presented below are the average over 20 runs.

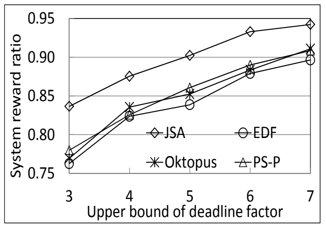
1) Effect of the number of jobs: To study the effect of the number of jobs on the system reward, in the simulation we randomly selected five sets of jobs from the trace, with total number of jobs in the set being 500, 1000, 1500, 2000 and 2500, respectively. In the real cluster, we ran five sets of jobs in the cluster, with total number of jobs in the set being 50, 100, 150, 200 and 250, respectively. Each set of jobs forms of 5 batches of jobs. Figures 4(a) and 5(a) show the system reward ratio for different scheduling methods, with respect to the number of jobs to schedule in the trace-driven simulation and real cluster implementation. We see that in both simulation and implementation, the result follows EDF<Oktopus≈PS-P<JSA. In both simulation and implementation, JSA performs much better than PS-P, Oktopus and EDF. This is because some jobs cannot receive enough bandwidth in PS-P, Oktopus and EDF, and hence they cannot be completed by their deadlines. JSA takes advantage the elastic feature of bandwidth



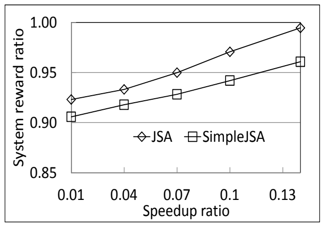
(a) System reward ratio vs. the number of jobs.



(b) System reward ratio vs. the average number of VMs of each job.

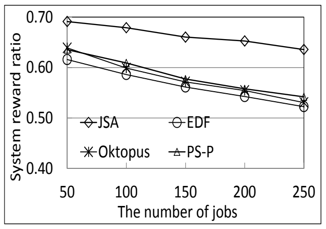


(c) System reward ratio vs. the upper bound of deadline factor.

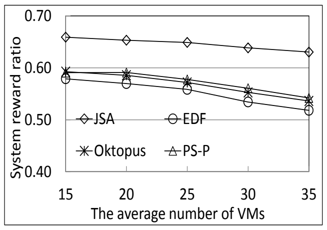


(d) System reward ratio vs. the speedup ratio.

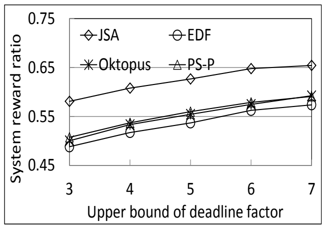
Fig. 4: Trace-driven simulation results.



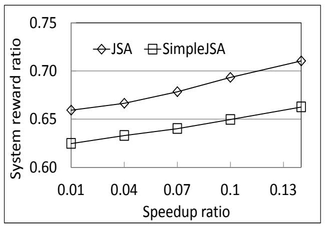
(a) System reward ratio vs. the number of jobs.



(b) System reward ratio vs. the average number of VMs of each job.



(c) System reward ratio vs. the upper bound of deadline factor.



(d) System reward ratio vs. the speedup ratio.

Fig. 5: Real cluster implementation results.

resources by using the minimum bandwidth option that meets its deadline in scheduling first and then tries to increase the reserved bandwidth. Also, by reducing the execution times of jobs in a batch, it leaves more bandwidth for scheduling for the subsequent batch, enabling more jobs to complete by their deadlines. As a result, JSA produces higher system reward ratio than PS-P, Oktopus and EDF. PS-P, Oktopus and EDF perform similarly.

2) Effect of the number of VMs per job: To investigate the effect of the number of VMs required by each job, we manually divided the jobs into five groups in the trace-driven simulation and real cluster implementation. In each group, the average number of VMs (N_j or tasks) of the jobs is 15, 20, 25, 30 and 35, respectively. Each group of jobs are divided into 5 batches of jobs as mentioned above. Figures 4(b) and 5(b) show the system reward ratio for each group in each job scheduling method in the trace-driven simulation and real cluster implementation, respectively. We see that the result follows EDF<Oktopus≈PS-P<JSA due to the same reasons as in Figure 4(a). We also see that the system reward ratio decreases as the average number of VMs of each job increases. The reason is also the limited capacity of the cloud. Due to a larger average size of virtual networks for jobs, jobs in the queue must wait for a longer time, thus generating more jobs that miss their deadlines.

3) Effect of the job deadlines: The workload model defines the job deadline as a multiple of the job execution time under its measured VM bandwidth $\bar{B}$. The factor β was randomly chosen from a range [2 ,7]. In this study, we still used 2 as its low bound and changed its upper bound to 3, 4, 5, 6 and 7, respectively. Figures 4(c) and 5(c) show the system reward ratio for each upper bound of each scheduling method in the trace-driven simulation and real cluster implementation, respectively. The curves in the figure have the same relationship as in previous figures and JSA achieves the most reward due to the same reasons. We also see that as the upper bound of the deadline factor increases, the system reward radio increases gradually. A larger upper bound of the factor increases the job deadlines such that the system has a better chance to receive reward from the jobs with larger deadlines.

4) Effect of the speedup ratio α_j: We use SimpleJSA to denote JSA without the job execution time optimization step of JSA (line 21-33 in Algorithm 1). This experiment is to see the effect of this optimization step and the effect of the speedup ratio α_j. Different cloud applications may have different speedup ratios with the bandwidth, and data-intensive applications with large network demands are expected to have higher speedup ratios. Figures 4(d) and 5(d) show the system reward ratio of SimpleJSA and JSA versus different α_j values in the trace-driven simulation and real cluster implementation, respectively. We see that JSA produces higher system reward ratio than SimpleJSA. The optimization step reduces the execution time of the jobs and allows the resources to release earlier. As a result, the jobs in the subsequent batch can be scheduled earlier, and hence have higher probability to meet their deadline, which leads to a higher system reward ratio. job execution time under a given bandwidth.

VI. Related Work

Currently, there exists very little work on the job scheduling within the context of bandwidth reservations for VM communication in clouds. In this section, we describe the related works on two topics: bandwidth allocation and job scheduling.
Bandwidth allocation. Oktopus [7] is a “hose model” based virtual cluster abstraction for network reservation, in which a virtual cluster request $<N, B>$ requires a virtual topology comprising of N machines connected by links of bandwidth B to a virtual switch. In these works, the bandwidth demand is specified in the request from the tenant. TIVC [8] further extends the virtual cluster model with time-varying bandwidth

reservation in order to capture the time-varying network demand of cloud applications. CloudMirror [23] proposes a network abstraction and an efficient VM placement strategy to provide predictable performance for cloud datacenter networks. FairCloud [20] presents three allocation policies to navigate the tradeoffs between minimum guarantee, network proportionality, and high utilization. Shen *et al.* [24, 25] proposed a new pricing model in order to achieve a win-win situation for cloud provider and tenants. Unlike the above works that allocate static bandwidth demands, our work utilizes the elastic feature of bandwidth resource and elastically reserves bandwidth for the jobs based on their deadline.

Job scheduling. Our work is related to the work in [9], which also addresses the optimal scheduling problem with the goal of maximizing value-based objective function. However, this work only considers the constraint of the number of CPU units. Several works [26, 27] propose job scheduling algorithm to improve the cluster performance. Corral [13] improves job performance and reduces network contention from rack to core switches by considering the workload of recurring jobs and jointly optimizing the input data placement and tasks. Unlike these job scheduling works, we focus on scheduling the type of job requests arising from the recently proposed virtual network abstraction. It is suitable for scheduling jobs comprised of communication VMs and can ensure sufficient bandwidth provision for jobs. Previous works use fixed bandwidth amounts for jobs in bandwidth reservation or job scheduling. The novelty of this work is that it takes advantage of the elastic feature of bandwidth to dynamically determine the bandwidth demands of jobs to achieve a certain objective such as maximizing value-based objective function.

VII. Conclusion

In this paper, we design a new cloud service model, in which a tenant only needs to specify the job deadline. We aim to find a job schedule (the reserved bandwidth in the virtual cluster abstraction, the running start time of each job, and the VM placement in the datacenter) to satisfy the deadline requirements while maximizing the total rewards of jobs that are finished by their deadlines. Due to the NP hardness of this problem, we propose a heuristic scheduling algorithm. We produce job profiles for jobs indicating the job execution time corresponding to each provisioned bandwidth amount. The algorithm first uses the minimum bandwidth amount of each job that meets its deadline in scheduling, and then increases the reserved bandwidth as much as possible to reduce the execution times of jobs. Our trace-driven simulation and real cluster implementation results show the efficiency and effectiveness of our algorithm in comparison with other scheduling algorithms. In the future, we will explore on-line scheduling algorithms with automatic bandwidth reservation.

VIII. Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145 and Microsoft Research Faculty Fellowship 8300751.

References

[1] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proc. of VLDB Endow.*, vol. 3, no. 1-2, Sep. 2010.

[2] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proc. of ODSI*, 2010.

[3] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. of NSDI*, 2011.

[4] Y. Lin and H. Shen, "Eafr: An energy-efficient adaptive file replication system in data-intensive clusters," in *Proc. of ICCCN*, 2015.

[5] 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 *Proc. of SIGCOMM*, 2013.

[6] 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 *Proc. of Co-NEXT*, 2010.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. of SIGCOMM*, 2011.

[8] D. Xie, N. Ding, Y. C. Hu, and R. R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *Proc. of SIGCOMM*, 2012.

[9] N. Jain, I. Menache, J. Naor, and J. Yaniv, "Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters," in *Proc. of SPAA*, 2012.

[10] G. Liu, H. Shen, and H. Wang, "Computing load aware and long-view load balancing for cluster storage systems," in *Proc. of Big Data*, 2015.

[11] M. Drozdowski, *Scheduling for Parallel Processing*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[12] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan, "Choreo: Network-aware task placement for cloud applications," in *Proc. of IMC*, 2013.

[13] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proc. of SIGCOMM*, 2015.

[14] B. Palanisamy, A. Singh, L. Liu, and B. Langston, "Cura: A cost-optimized model for mapreduce in a cloud," in *Proc. of IPDPS*, 2013.

[15] L. Chen and H. Shen, "Consolidating complementary vms with spatial/temporal-awareness in cloud datacenters," in *Proc. of INFOCOM*. IEEE, 2014, pp. 1033–1041.

[16] H. Herodotou, F. Dong, and S. Babu, "Mapreduce programming and cost-based optimization? crossing this chasm with starfish," *PVLDB*, vol. 4, no. 12, pp. 1446–1449, 2011.

[17] L. Yan and H. Shen, "TOP: vehicle trajectory based driving speed optimization strategy for travel time minimization and road congestion avoidance," in *Proc. of MASS*, 2016.

[18] 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 *Proc. of SIGCOMM*, 2009.

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

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

[21] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance Using Workload Suites." in *Proc. of MASCOTS*, 2011.

[22] "CloudLab cluster," http://https://www.cloudlab.us/.

[23] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *Proc. of SIGCOMM*, 2014.

[24] H. Shen and Z. Li, "New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants," in *Proc. of INFOCOM*, 2014.

[25] ——, "New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2015.

[26] H. Shen, A. Sarker, L. Yu, and F. Deng, "Probabilistic network-aware task placement for mapreduce scheduling," in *Proc. of Cluster*, 2016.

[27] J. Liu and H. Shen, "Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds," in *Proc. of CloudCom*, 2016.