DCloud: Deadline-aware Resource Allocation for Cloud Computing Jobs

Dan Li*, Congjie Chen*, Junjie Guan*, Ying Zhang†, Jing Zhu*, Ruozhou Yu*

*Tsinghua University †HP Labs

***Abstract*—With the tremendous growth of cloud computing, it is increasingly critical to provide quantifiable performance to tenants and to improve resource utilization for the cloud provider. Though many recent proposals focus on guaranteeing job performance (with a particular note on network bandwidth) in the cloud, they usually lack efficient utilization of cloud resource, or vice versa.**

In this paper we present DCloud, which leverages the (soft) deadlines of cloud computing jobs to enable flexible and efficient resource utilization in data centers. With the deadline requirement of a job guaranteed, DCloud employs both *time sliding* (postponing the launching time of a job) and *bandwidth scaling* (adjusting the bandwidth associated with VMs) in resource allocation, so as to better match the resource allocated to the job with the cloud's residual resource. Extensive simulations and testbed experiments show that DCloud can accept much more jobs than existing solutions, and significantly increase the cloud provider's revenue with less cost for individual tenants.

I. Introduction

The proliferation of virtualization technologies and rich Internet connectivity have brought an explosive growth in cloud computing [1], [2], [3]. Tenants outsource their computation and storage to public cloud providers, and pay for the service usage on demand. This model offers unprecedented advantages in terms of cost and reliability, compared with traditional computing model that uses dedicated, in-house infrastructure. Despite the tremendous momentums it grows, the future viability of cloud computing, however, still depends on its offered performance to tenants.

Amazon EC2: Amazon EC2 [1] is a representative public cloud service, it requires a tenant to specify the number (type) of virtual machines (VMs) needed, which are then assigned to available physical servers by the cloud provider. VMs establish TCP connections for data transfer. The bandwidth of a VM depends on the number of contending TCP flows along the path. As a result, the network performance of a tenant is not guaranteed at all, and a selfish tenant can get more bandwidths than others by simply establishing more TCP connections, even under the constraint of bandwidth capping mechanism [4]. It not only makes the task finish time of cloud computing jobs unpredictable, but also affects the cost a tenant pays, since tenants are charged based on the duration of VM occupation in EC2.

The work was supported by the National Key Basic Research Program of China (973 program) under Grant 2014CB347800, the National Natural Science Foundation of China under Grant No.61170291, No.61432002, the National High-tech R&D Program of China (863 program) under Grant 2013AA013303, and Tsinghua University Initiative Scientific Research Program.

Hard Bandwidth Guarantee: In order to achieve predictable performance for cloud computing jobs, hard bandwidth guarantee is proposed to provide to tenants, represented by SecondNet [5], Oktopus [6], *etc.* In this kind of solutions, a tenant specifies both the number of VMs and associated bandwidth of each VM (for hose-model [7] bandwidth abstraction) or VM pair (for matrix-model bandwidth abstraction). Cloud provider then guarantees the tenant's bandwidth requirement by allocating the reserving bandwidth in related links. Although the application performance is predictable by this way, the cloud resource may not be efficiently utilized for two reasons. First, bandwidth fragmentation will naturally exist during bandwidth allocation. Second, the VM and bandwidth requests specified by the tenants may not well match the residual resource in the cloud. The works in [8] and [9] then improve the bandwidth utilization by considering variable bandwidth requirements across time and across VMs, respectively. But both solutions still require the tenants to specify their resource demands, and thus cannot fundamentally solve the problem. Bazaar [10] tries to adjust the number of VMs and the associated bandwidths allocated to a tenant, so as to better match the VM and networking resource in the cloud. However, the solution is designed specifically for MapReduce jobs. If considering a broader range of cloud applications, arbitrarily varying the number of VMs may break the application semantics.

Minimum Bandwidth Guarantee: To improve bandwidth utilization besides providing predictable network performance, some very recent works propose minimum bandwidth guarantee to tenants, while letting the residual bandwidth shared by tenants in a best-effort manner [11], [13], [12]. By this way, the network performance a tenant gets is no worse than the minimum guaranteed bandwidth, and is thus predictable. Meanwhile, the available bandwidth can be fully utilized by tenants with higher traffic demands. Although this approach significantly improves bandwidth utilization in the cloud, they may not be able to meet the demand of cloud computing jobs with deadline requirements. The reason is obvious: the minimum bandwidth guaranteed may be less than the bandwidth required by a job to meet its deadline, while there is no guarantee on the share of extra bandwidth. To guarantee jobs

Bandwidth allocation solution	Efficient resource utilization	Predictable network performance	Guaranteeing Job deadline
EC2 [1]	✓	✗	✗
Hard bandwidth guarantee (Oktopus [6], TIVC [8], *etc.*)	✗	✓	✓
Minimum bandwidth guarantee (FairCloud [11], ElasticSwitch [12], EyeQ [13], *etc.*)	✓	✓	✗
DCloud	✓	✓	✓

TABLE I
COMPARISON AMONG DCLOUD AND OTHER BANDWIDTH SHARING SOLUTIONS.

finishing before deadlines, tenants need to accurately set the minimum bandwidth the job needs, which is a great burden for them.

Our Work: In this work we propose DCloud, which is a new interface between tenants and provider for cloud computing with deadline requirements. In today's cloud infrastructure, we have found that data analytic jobs account for a large proportion of cloud jobs, such as web logs analysis, weather forecast analysis, finance analysis, scientific simulation, machine learning, etc [14], [15]. A great part of these jobs have deadline requirements since results of them may be useless if they do not finish in time. Cloud resource allocation for these jobs are the focus of DCloud. DCloud requires a tenant to specify both the required resource and the *job deadline* when submitting a job request to the cloud. The required resource is quantified by the number of VMs and associated bandwidth, as well as the referenced *job running duration* profiled under the requested VMs and bandwidths. When the cloud provider allocates the resource, she can leverage the time interval between the job running duration and the job deadline to reshape the resource request, which leaves room to efficiently utilize the residual cloud resource without violating the job's deadline. We develop a novel resource allocation algorithm to exploit the room, which uses *time sliding* to smooth out the peak demand and *bandwidth scaling* to balance the usage of network resource and non-network resource in the cloud. Table. I compares DCloud with other bandwidth sharing solutions.

Although the concepts above are intuitive and promising, many challenges exist in transferring the basic ideas into a practical system. We employ a number of mechanisms and algorithms to address the challenges. First, when reshaping the tenants' requests, we depend on an *inversely proportional* rule to conservatively estimate the job running duration after bandwidth scaling, without knowledge of the application semantics. Second, we use the metric of *dominant resource utilization* to perform joint optimization of different types of cloud resources when allocating them to the tenants. Thirdly, we introduce a *profiling relax index* to mask the possible profiling errors from the tenants. Finally, we design a strategy-proof and job-based charging mechanism to encourage tenants to submit true deadline and resources.

We conduct extensive simulations and testbed experiments to study the performance of DCloud, with comparisons with the current model (e.g. Amazon EC2) and the recently proposed virtual cluster (VC) abstraction. The results show that DCloud can significantly improve the VM and bandwidth utilization under various settings and complete more than twice jobs within deadlines. In spite of less cost for individual job, DCloud can increase the cloud provider's revenue by more than 50%.

The rest of this paper is organized as follows. Section II and Section III present the design overview and details of DCloud, respectively. Section IV and Section V evaluate DCloud by extensive simulations and testbed experiments, respectively. Section VI discusses the limitations of DCloud and possible solutions. Section VII introduces related works. Finally, Section VIII concludes the paper.

II. DESIGN OVERVIEW

In order to both offer cloud providers the flexibility in allocating the resource to tenants and meet the deadlines of cloud computing jobs, DCloud requires each tenant to explicitly express the expected job deadline as well as the resource required (specified by the number of VMs, the associated bandwidth for each VM, and the profiled job running duration). Depending on tenant's requirement, the job deadline may be larger than the profiled job running duration with the requested VMs and bandwidths, leaving room for cloud provider to flexibly decide the launching time and assigned bandwidth for the job. Besides, a job-based and strategy-proof charging mechanism is designed to encourage tenants to submit the actual resource request, which favors the resource allocation algorithm in efficiently utilizing the cloud resource. DCloud contains three major modules to be described below, along with key notations in Table. II.

Resource Request: A tenant's request is described as a 4-tuple $J =< n, b, p, d >$ in DCloud. Similar as previous abstraction models, n and b are the number of VMs and the bandwidth needed per VM. Note that b can be varied across different VMs [9] or over time [8]. For simplicity, we only focus on the *homogeneous* bandwidth scenario in this paper, *i.e.,* for a specific job, b has the same value for all its VMs constantly over time. But our work can also be extended to embrace heterogeneous bandwidth requirements, and we leave it as future work. The novelty in DCloud's request abstraction is the last two variables: the job running duration, p, and the job deadline, d. Naturally, a tenant has the expected latest time for the job to finish, after which the result may not be usable. d is expressed as the interval between the current time and

Notations	Definitions
J	A job request
J'	The relaxed form of a job request
Y	The reshaped form of a job request
n	The requested number of VMs
b	The requested bandwidth for each VM
b'	The scaled bandwidth for each VM
t'	The interval between the submitting time and the launching time
p	Job running duration
$\tilde{p}$	Job running duration after relaxation
p'	The new job running duration under b'
d	The interval between the submitting time and the expected latest time

TABLE II
KEY NOTATIONS IN DCLOUD.

the expected latest time. p can be obtained by profiling the application with a small sample input [8], [16], [17], [18].

Both the provider and tenants can benefit from the knowledge of p and d. On the one hand, the tenant gets the performance guarantee as the job will be completed no later than the expected time. On the other hand, the provider does not have to launch the job immediately or always strictly provide b amount of bandwidth to the tenant. Instead, the provider can make more flexible and efficient allocation of the cloud resource, by leveraging the relationship among b, p and d.

Resource Allocation: Benefiting from the additional information in the request abstraction, we design a new allocation algorithm based on two mechanisms, *i.e., time sliding* and *bandwidth scaling*. By time sliding, we allow a lag between the job submitting time and the actual launching time. Hence we can smooth out the peak demand of the cloud, and reduce the number of rejected tenants at busy time. By bandwidth scaling, we allow dynamic adjustment of the bandwidth allocated to VMs. Since bandwidth is usually the bottleneck resource for applications run in the cloud [19], [20], in this paper we only consider *scaling down* the requested bandwidth rather than *scaling up*, by which we can achieve more balanced utilization of different types of cloud resource. As a whole, DCloud reshapes the resource requests in both temporal dimension and spatial dimension to improve the resource utilization.

Charging: The information in the resource request tuple is the key input in our allocation. However, if a selfish tenant always declares the job deadline as small as the job running duration, or intentionally requests a very small bandwidth that reversely change the balance between VM demand and bandwidth demand, the advantage of DCloud's resource allocation algorithm in efficiently utilizing the cloud resource will be limited. We thus design a job-based and strategy-proof charging mechanism for DCloud, which encourages tenants to truthfully declare b, p and d to minimize their costs.

III. DCLOUD DESIGN

A. Reshaping the Resource Requests

When receiving a job request $J =< n, b, p, d >$ from a tenant, DCloud provider takes two steps to reshape it. First, the resource request is relaxed to $J' =< n, b, \tilde{p}, d >$ to mask the possible profiling error, where $\tilde{p}$ is the relaxed form of p. Second, the relaxed request tuple $J' =< n, b, \tilde{p}, d >$ is reshaped to $Y =< n, b', t', p' >$ to better fit into the available cloud resource. Here, b' is the scaled VM bandwidth, t' is the interval between the submitting time and launching time of the job, and p' is the new job running duration under b'. Y should satisfy $t' + p' \leq d$ to meet the job's deadline requirement. In this work we assume that n is determined by the application semantics and cannot be arbitrarily set. We will explore how to adapt n without breaking application semantics in future work.

To mask the possible profiling error, cloud provider first relaxes p to $\tilde{p} = \min((1 + \gamma) \times p, d)$, where γ is called the *profiling relax index*. The relax of p to $\tilde{p}$ helps guarantee the job completion if p is underestimated in profiling, since the cloud resource beyond the expected job finish time will be reserved for other jobs. But $\tilde{p}$ should be no larger than d. Note that the conservatively reserved resource will be released to serve other jobs immediately after the target job is completed. γ is a parameter configured by specific cloud provider, and can be regarded as the cloud provider's *maximum tolerable underestimation* for the job running duration in a tenant's request. We will evaluate the impact of γ on resource allocation in Section IV.

Second, cloud provider uses time sliding and bandwidth scaling to reshape the relaxed request tuple $J' =< n, b, \tilde{p}, d >$ to $Y =< n, b', t', p' >$. Time sliding decides the launching time of the job (t'), while bandwidth scaling determines the actual bandwidth assigned to each VM (b') and the new job running duration (p') under the scaled bandwidth. We will discuss the detail about how to determine the value of t' and b' in Section III-D.

The major benefit of time sliding is to smooth out the peak demand of the cloud, and thus reduce the number of rejected tenants at busy time. If the gap between the job duration and the deadline is long enough, we could even shift some day-time jobs to be run at night time, given the diurnal pattern of cloud loads [21], [22]. It not only improves the peak time experience, but also increases the valley time resource utilization.

The purpose of introducing bandwidth scaling, *i.e.,* adjusting bandwidth request b in J' to a smaller b' in Y, is to balance the utilization of VM and network resource in the cloud, as network bandwidth is found to be the bottleneck resource for applications run in the cloud data centers [19]. Bandwidth scaling benefits efficient utilization of the cloud resource in two aspects. First, intuitively, it allows acceptance of more tenants, when the available bandwidth in the cloud is not sufficient to meet a tenant's original bandwidth demand while the VM slots are plenty. Second, it can help allocate

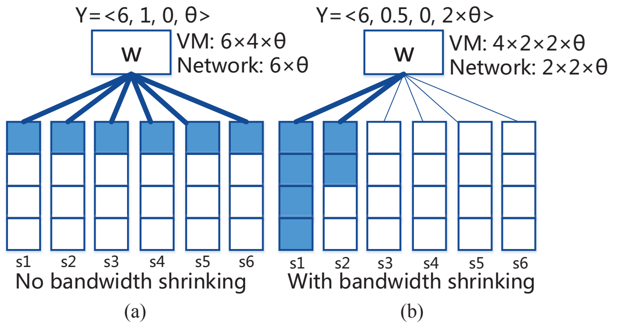


Fig. 1. Benefit of bandwidth scaling in saving network resource by localized VM allocation. The relaxed request is $J' =< 6, 1, \theta, 2 \times \theta >$. Y indicates the reshaped request based on J'.

VMs in a more localized way. Assigning VMs of one tenant to physically close-by servers will reduce the total bandwidth consumption in the network. We will describe how to estimate the prolonged job running duration p' after scaling down the bandwidth in Section III-B.

We illustrate the second benefit using an example in Fig. 1. In this network topology, a switch is connected to 6 servers, each of which contains 4 VM slots and the bandwidth of each link is 1 unit. Now a tenant submits a request and it is relaxed as $J' =< 6, 1, \theta, 2 \times \theta >$, *i.e.,* asking for 6 VMs with 1 unit bandwidth each, job running duration of θ, and the job deadline of $2 \times \theta$. Without bandwidth scaling, the only allocation method is to assign one VM per server, as is shown by the shaded boxes in Fig. 1(a). Since each VM uses up the 1 unit bandwidth of the server's link, it is not possible to accept any other tenants (with more than 3 VMs) in the next time interval of θ. Therefore, practically this tenant consumes $24 \times \theta$ VM time and $6 \times \theta$ bandwidth time.

However, if we scale down the requested bandwidth to 0.5 unit, we have a more efficient allocation shown as Fig. 1(b). Note that the largest bandwidth needed between the two assigned servers is $2 \times 0.5 = 1$, since the second server has only two VMs. As a tradeoff, the conservatively estimated job running duration will be lengthened to $2 \times \theta$. In this way, the total resource consumed by this tenant is at most $8 \times 2 \times \theta = 16 \times \theta$ VM time and $2 \times 2 \times \theta = 4 \times \theta$ bandwidth time, respectively. Compared with Fig. 1(a), it reduces the resource consumption by one third. Nevertheless, we should avoid over-scaling of bandwidth, since we need to make high utilization of the network resource as well.

B. Estimating the Job Finish Time

A major challenge in bandwidth scaling is to estimate the new job running duration, p', after scaling down the bandwidth associated to VMs. To make the resource allocation algorithm scalable, we assume that the cloud provider has no knowledge about the application semantics, and thus we have to make the estimation based on the input parameters in resource request $J' =< n, b, \tilde{p}, d >$ alone. We develop a simple yet effective method for the estimation.

We divide the resources a job needs into two parts, namely, network bandwidth and non-network resource (VM's local resources, *e.g.,* CPU, memory, local disk). Firstly, we suppose the running duration of a job is τ given that the network bandwidth is never constrained, *i.e.,* the bandwidth is infinite. Under this setup, the bandwidth demand of the job when the job is completed by the percentage of x is denoted as $c(x)$. In practice the bandwidth resource should be finite and subject to a time-varying function $b(t)$ (both $c(x)$ and $b(t)$ are integrable). If at time t the job is completed by the percentage of x, the job running speed on the overall progress is:

$$\frac{dt}{dx} = \begin{cases} \frac{c(x)}{b(t)} \cdot \tau & c(x) > b(t) \quad \text{Network constrained;} \\ \tau & c(x) \le b(t) \quad \text{Otherwise.} \end{cases} \tag{1}$$

In DCloud we have to estimate the prolonged job running duration p' after scaling the bandwidth down to b' based on the known information of $\tilde{p}$ and b. Since we have no knowledge of τ or $c(x)$, we are not aware whether networking bandwidth or non-network resource is the constrained resource at a specific time. Hence, we can only estimate the upper bound of the job running duration, which is the worst case by assuming network bandwidth is always the constrained resource. If in real case the non-network resource is the constrained one in some time intervals, the actual job running duration should be less than the estimated. By estimating the upper bound of job running duration, we can reserve enough resource for a job.

Based on Eq. 1, we have

$$\tilde{p} = \tau \int_0^1 \max\left\{1, \frac{c(x)}{b}\right\} dx$$

and

$$p' = \tau \int_0^1 \max\left\{1, \frac{c(x)}{b'}\right\} dx$$

Therefore, we get

$$\begin{aligned} p' &= \frac{\int_0^1 \max\left\{1, \frac{c(x)}{b'}\right\} dx}{\int_0^1 \max\left\{1, \frac{c(x)}{b}\right\} dx} \cdot \tilde{p} \\ &\le \max_{x\in[0,1]} \left(\frac{\max\left\{1, \frac{c(x)}{b'}\right\}}{\max\left\{1, \frac{c(x)}{b}\right\}} \right) \cdot \tilde{p} \\ &\le \frac{b}{b'} \cdot \tilde{p} \end{aligned}$$

The worst case can be reached when $c(x) \ge b$ for $x \in [0, 1]$. We always use $p' = \frac{\tilde{p} \times b}{b'}$ to estimate the job running duration when allocating and reserving cloud resource for the job, and we call it the *inversely proportional rule*. Fig. 2 shows an example in which the bandwidth is scaled from b to $\frac{b}{2}$, and thus the estimated job running duration doubles. We will further validate the hypothesis in Section V. It is worth noting that if the job finishes earlier than the expected time because of the conservative estimation, the reserved resource is immediately released to serve subsequent jobs.

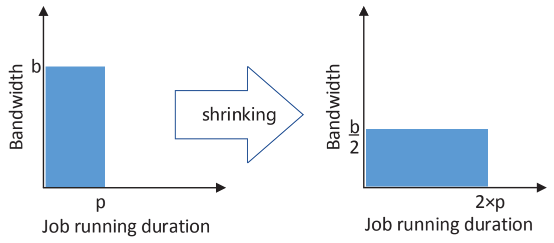
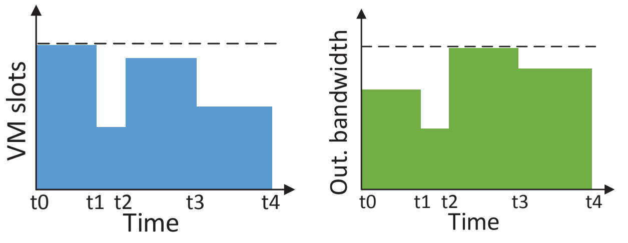


Fig. 2. Example of bandwidth scaling.



(a) VM utilization on a server (b) Bandwidth utilization on a link

Fig. 3. Time-varying resource utilization record for a server and a link.

C. Optimizing the Cloud Resource Utilization

With the merits of time sliding and bandwidth scaling in efficiently utilizing the cloud resource, we still need to determine the best t' and b' used in Y. In order to optimize the utilization of cloud resource in terms of different types of resources, particularly, VM slot (including CPU, memory and local disk for the VM slot) and link bandwidth, we propose a metric called *dominant resource utilization* (DRU) to compare different assignments of t' and b'. DRU captures the capacity of the cloud in accepting future requests, depending on its more stringent resource types. The goal of our resource allocation algorithm is to minimize the DRU of the cloud.

We assume a tree topology is used in the cloud network, and recursively define DRU for a server, a switch and the network as follows. For a server i, we assume it has m_i VM slots in total and there are already $m'_{i,t}$ slots occupied at time t. These occupied VMs consume $z'_{i,t}$ bandwidth out of the z_i total capacity of the server's outbound link. The DRU of the server at time t is then shown in Eq. 2, which takes the maximum from the utilization of VM resource (the first field) and the utilization of outbound link resource (the second field).

$$u_{i,t} = \max(\frac{m'_{i,t}}{m_i}, \frac{z'_{i,t}}{z_i}) \tag{2}$$

For a switch i, we assume that the set of all its children nodes is C_i, and the DRU for a child node $j \in C_i$ at time t is $u_{j,t}$. The outbound link capacity of switch i is z_i, among which $z'_{i,t}$ bandwidth has been utilized at time t. Shown in Eq. 3, the DRU of switch i at time t is either its outbound link bandwidth utilization or its children's average DRU, whichever is larger.

$$u_{i,t} = \max(\frac{\sum_{j\in C_i} u_{j,t}}{|C_i|}, \frac{z'_{i,t}}{z_i}) \tag{3}$$

After defining the DRU for a node at each time instance, the DRU of the node is the integral over time, as illustrated by Eq. 4. This is because resource occupation time is also an important factor that affects the cloud resource utilization.

$$u_i = \int_{t=0}^{\infty} u_{i,t} \tag{4}$$

Based on the definition, the larger a node's DRU is, the more difficult it is to accept subsequent tenant requests within the subtree rooting from the node. When defining the DRU of the network, we take the network oversubscription in typical tree topology into account. Since higher-level nodes and links in the tree are easier to become congested, we separate three types of network's DRUs, *i.e., aggregation-layer DRU* (denoted as u_1), *access-layer DRU* (denoted as u_2) and *server-layer DRU* (denoted as u_3), which are the sum of the DRUs of all the aggregation-layer switches, access-layer switches and servers, respectively. We do not consider the core-layer switch, because its DRU is just the aggregation-layer DRU divided by the number of aggregation switches. The DRU of the cloud network, U, is then defined as the tuple of $U =< u_1, u_2, u_3 >$. The value of U is larger if a former variable in the tuple is larger.

We again use Fig. 1, which is part of a three-layer tree structure, as an example to illustrate how to calculate DRU. In Fig. 1(a) without any bandwidth scaling, the DRU of each server s_i $(0 \le i < 6)$ is $u_{s_i} = \int_{t=0}^{\theta} \max(0.25, 1) = \theta$, hence the DRU of the switch w is $u_w = \max(\frac{\sum_{i=0}^{5} u_{s_i}}{6}, 0) = \theta$. But for Fig. 1(b), we have $u_{s_0} = \int_{t=0}^{2\times\theta} \max(1,1) = 2\times\theta$, $u_{s_1} = \int_{t=0}^{2\times\theta} \max(0.5,1) = 2\times\theta$, and $u_{s_i} = 0$ $(2 \le i < 6)$, hence the DRU of switch w is $u_w = \max(\frac{\sum_{i=0}^{5} u_{s_i}}{6}, 0) = \frac{2\times\theta}{3}$. Since the DRU of the other parts of the network are not affected by the resource allocation under switch w, the DRU of the network with $b' = 0.5$ is smaller than that with $b' = 1$. Consequently, $b' = 0.5$ is a better choice to reshape the request.

D. Resource Allocation Algorithm

In what follows we present our resource allocation algorithm in DCloud, with a particular emphasis on how to select t' and b' in order to minimize the DRU of the network. Previously proposed resource allocation algorithms [6], [8], [9] assign VMs and bandwidth for a job request based on the *instantaneous resource utilization state* (RUS), for which we use the function *VCalloc*(n,b,V) to express, where V denotes the instantaneous RUS. In DCloud, however, we need to determine the launching time of each job as well, thus we propose a new abstraction called *time-varying resource utilization record* (RUR), denoted by R, to maintain the VM utilization for every server and bandwidth utilization for every link over time. We implement RUR as an array of link lists. The bar plotted in Fig. 3(a) and Fig. 3(b) are used to visually demonstrate the time-varying RUR for a server and a link,

Algorithm 1: DCloud resource allocation algorithm.

```
1  DCloudAlloc($J', R$)
   Input: $J' =< n, b, \tilde{p}, d >$: relaxed resource request;
   $R$: existing time-varying RUR;
   $R_{o,t}$: utilized resource for server/link $o$ at time $t$.
   Output: $R$: updated time-varying RUR.
2  $U \leftarrow< \infty, \infty, \infty >$;
3  for $t'$ from 0 to $(d - \tilde{p})$ with step $\frac{d-\tilde{p}}{10}$ do
4      for $b'$ from $b$ downto $\frac{b\times\tilde{p}}{d-t'}$ with step $\frac{b\times(d-t'-\tilde{p})}{10\times(d-t')}$ do
5          $p' \leftarrow \frac{\tilde{p}\times b}{b'}$;
6          $Y' \leftarrow< n, b', t', p' >$;
7          if Place($Y', R$)=$True$ then
8              $U' \leftarrow< u_1, u_2, u_3 >$; /*network's DRU*/
9              if $U' < U$ then
10                 $U \leftarrow U'$;
11                 $Y \leftarrow Y'$;
12             end
13         end
14     end
15 end
16 $R \leftarrow$ updated time-varying RUR with $Y$;
17 ————————————————————————————
18 Place($Y', R$)
   Input: $Y' =< n, b', t', p' >$: reshaped resource request;
   $R$: existing RUR.
   Output: $True$ if successfully allocated, $False$ otherwise.
19 foreach server or link $o$ in the cloud do
20     $V_o \leftarrow \max_{t\in[t',t'+p']} R_{o,t}$; /*turning RUR to RUS*/
21 end
22 return VCalloc($n, b', V$);
```

respectively. It is updated when a job is allocated or when a job is finished before the expected ending time.

Algorithm 1 describes the resource allocation procedure in DCloud. We separate the continuous search space for t' and b' to discrete bins, and use a simple enumeration approach to find the best assignments. More specifically, we start by enumerating the possible starting time t', which can be any value from the request submission time ($t' = 0$), up to the latest starting time that the job can finish right at its deadline ($t' = d - \tilde{p}$). We separate values in this range into x bins, where x is a configurable variable (we use $x = 10$ in line 3 of Algorithm 1). Similarly, the range for bandwidth scaling is from its requested value b, down to the smallest value by which the job running duration will be prolonged to right before the deadline ($\frac{b\times\tilde{p}}{d-t'}$). This range is also separated into bins (we use 10 bins in line 4 of Algorithm 1). Among all the options, we find the best allocation that minimizes DRU of the network, U (lines 7-13).

When assigning VM and bandwidth with a given b' and t', we translate the time-varying resource tracked in R to the instantaneous resource in V. It is done by taking the maximum utilized resource during the time interval of $[t', t'+p']$ for every server and every link (line 20). Then we use *VCalloc*(n,b',V) (line 22) to assign the VMs and link bandwidth. After choosing the b' and t' that can minimize the DRU of the network, we need to update R, by adding the reserved VM slots and bandwidth from t' to $t' + p'$ onto the corresponding servers and links (line 16). When a job finishes at $t'' < t' + p'$, the resource reserved at the corresponding servers and links in the time interval of $[t'', t' + p']$ are released to serve subsequent pending jobs.

E. Charging

Although DCloud's resource allocation algorithm can work for any forms of resource requests from tenants, its advantage is particularly obvious when the following two assumptions hold. First, there is an interval between the job finish time and tenant's latest tolerable deadline. Second, network is the bottleneck resource in the cloud; or in other words, real applications' bandwidth demands are higher than that the cloud network can provide. The former should hold for most applications in reality, which favors the effectiveness of both time sliding and bandwidth scaling; while the latter is proven by existing cloud data centers [19], [23], which is the original motivation of bandwidth scaling. However, strategic tenants may deliberately break the two assumptions in their resource requests as follows, if there is **no** mutual trust between tenants and cloud provider.

First, a tenant can cheat by requesting unnecessarily more resource, *e.g.,* larger n, b or p than the actually required, or by declaring a shorter job deadline than the actual one, *e.g.,* setting d the same as p. Either one can break the first assumption. Second, a tenant requests the actually required resource without cheating, but uses a very small b to profile the application, *e.g.,* setting b small enough to prolong p to the actual deadline d. This kind of skewed resource requests may break both the assumptions above.

DCloud leverages its *strategy-proof* charging mechanism to overcome the problems and encourage selfish tenants to submit the actual resource requests. In addition, the charging should be *job-based*, indicating that a tenant is charged based on the resource request tuple $J =< n, b, p, d >$ alone, regardless of the specific resource allocation policy taken by the cloud provider.

Specifically, supposing $C = F(.)$ as the charging function, it should satisfy the following requirements to meet the properties of the charging mechanism.

1) $C = F(n, b, p, d)$. $F(.)$ does not depend on the actually allocated resource in the cloud, *e.g.,* b', t', p', *etc.*
2) Given $n_1 < n_2$, $b_1 < b_2$, $p_1 < p_2$ and $d_1 < d_2$, there should be $F(n_1, b, p, d) < F(n_2, b, p, d)$, $F(n, b_1, p, d) < F(n, b_2, p, d)$, $F(n, b, p_1, d) < F(n, b, p_2, d)$ and $F(n, b, p, d_1) > F(n, b, p, d_2)$, respectively. Hence, a tenant has to pay more, if cheating with unnecessarily more resource than actually required or shorter deadline than the actual one.
3) Supposing b_s is the maximum speed an application can generate traffic without bandwidth limit, and p_s is the profiled job running duration with bandwidth b_s, $F(n, b, p, d)$ is minimized when $b = b_s$ and $p = p_s$. Consequently, tenants will not use a very small b to profile the application and submit a skewed resource request.

Assuming α is the unit price for network bandwidth (\$/(bps·hour)) and β is the unit price for VM resource (\$/hour), we propose a general charging framework as Eq. 5. We call $g(\frac{d}{p})$ the *discount function*, which *monotonically decreases* over $\frac{d}{p}$, and equals to 1 when $d = p$. Besides, we call $\frac{d}{p}$ the *deadline extension ratio*, which is a measure of flexibility for request reshaping.

$$C = (\alpha \times b + \beta) \times n \times p \times g(\frac{d}{p}) \tag{5}$$

It is obvious that the charging function satisfies the requirement 1 above. By fixing other input parameters, $C = F(.)$ *monotonically increases* with larger b or p, and *monotonically decreases* with larger d. Consequently, the charging function also satisfies requirement 2.

Next we will show that this charging function also meets the last requirement. Note that when profiling applications, p and b are not independent. We first deduce the relationship between p and b. When $b \geq b_s$, where b_s is the maximum traffic rate generated by the application, p does not decrease with larger b, because the computation bottleneck for the job is not in network. Hence, $C = F(.)$ *monotonically increases* with larger b when $b \geq b_s$.

But when $b \leq b_s$, network bandwidth is the bottleneck resource for computation, and p is inversely proportional to b. Hence we have $p = \frac{Q}{b}$, where Q is the constant total amount of traffic a VM generates determined by application. Then the cost can be specified as

$$C = (\alpha \times b + \beta) \times n \times \frac{Q}{b} \times g(\frac{b \times d}{Q}), \textbf{when } b \leq b_s \tag{6}$$

By noting that $g(.)$ is a decreasing function over the input, Eq. 6 demonstrates that $C = F(.)$ *monotonically decreases* with larger b when $b \leq b_s$, by taking the dependence of p on b into consideration.

As a whole, the value of $C = F(.)$ is minimized when $b = b_s$ and thus the charging function of Eq. 5 can meet the requirement 3. A strategic tenant will always submit resource request with $b = b_s$, which favors the resource allocation algorithm in efficiently utilizing the cloud resource.

IV. SIMULATION

We develop an in-house event-driven simulator to evaluate the performance of DCloud, with particular comparison to two other representative resource allocation algorithms, namely, *baseline* (*BL*) allocation simulating the current model that do not provide bandwidth guarantee for tenants and *virtual cluster*(*VC*) algorithm used in Oktopus [6].

A. Simulation Setup

We use DCloud, BL and VC to allocate the cloud resource for requested jobs, respectively. In BL allocation, a lowest subtree is found to place VMs, so as to reduce the network traffic. VMs compete for the network bandwidth using TCP. A tenant is rejected only if there are no available instantaneous VM slots. Though BL is similar to the current model, we make a minor modification on it that a job is immediately killed if it does not finish at the job deadline. Hence, the BL allocation in our simulation can actually accept more jobs than the standard resource allocation algorithm used in the current model. In VC allocation, we use the algorithm in [6] to allocate VMs with bandwidth guarantee, and a tenant is rejected if there is no sufficient instantaneous bandwidth or available instantaneous VM slots. We simulate tenant's dynamical arrival over 24 hours, and separately use the three algorithms for resource allocation.

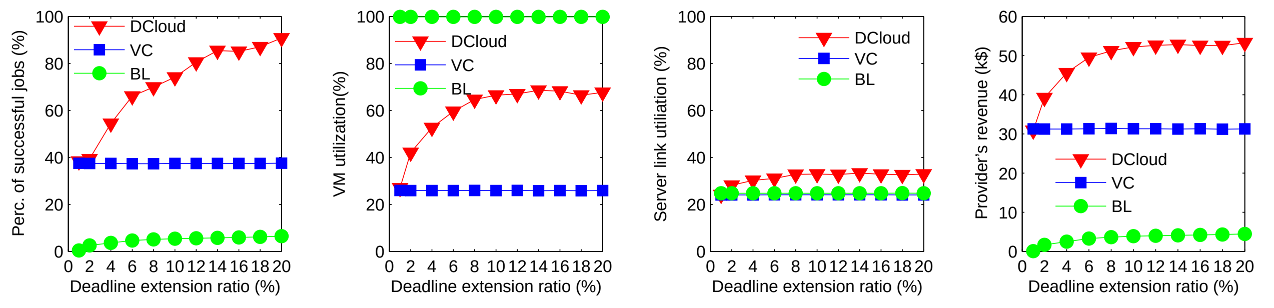
Comparison Metrics: We use four metrics to quantify the results.

- The percentage of successful jobs that meet deadlines out of all the job requests. The results exclude both the jobs that are accepted but fail to meet the deadlines, and the jobs that are rejected by the allocation algorithm at the beginning.
- The VM utilization in the cloud, *i.e.,* the average percentage of VMs being occupied across time.
- The server link utilization, *i.e.,* the average utilization of the server links over time, which captures the traffic amount sent by applications to the network.
- The total revenue of the cloud provider. We only charge the jobs that meet their deadlines. For DCloud, we use the charging function of $C = (\alpha \times b + \beta) \times n \times p \times \log_{10}(9 + \frac{p}{d})$. But for BL and VC, we do not consider the discount, *i.e.,* the charging function is $C = (\alpha \times b + \beta) \times n \times p$. In other words, for an individual successful job which meets the deadline, the charge in DCloud is lower than that in BL and VC. We set β =0.085\$/hour (the price for current Amazon EC2 small VMs) and α =0.288\$/(Gbps·hour), so the charges for bandwidth resource and VM resource are comparable. It is notable that in BL, since there is no performance guarantee when allocating the resource, the jobs which run for a while but miss their deadlines are counted to the cloud provider's revenue.

Topology and Parameters: We use a 3-layer 20-array tree structure as the cloud network topology. Specifically, the core switch connects to 20 aggregation switches, every aggregation switch connects to 20 ToR switches, and every ToR switch connects to 20 servers. Every server has 4 VM slots. So there are in total 8,000 servers and 32,000 VMs in the cloud network.

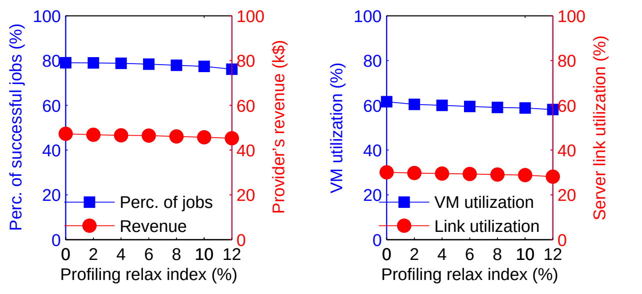
For the tree topology, the capacity of each link connecting servers and ToR switches is 1Gbps, and the capacities of upper-layer links are varied to study the impact of network oversubscription ratio, from 1:1 to 16:1. For instance, for the default oversubscription ratio of 8:1, the capacity of links connecting ToR switches and aggregation switches is set as 5Gbps, while that of links connecting aggregation switches and the core switch is 50Gbps.

We generate tenant requests $J =< n, b, p, d >$ using the following parameter settings. The number of VMs, n, is subject to the normal distribution of $Norm(\overline{n}, 25\overline{n}^2)$, with the mean of $\overline{n}$ and deviation of $25\overline{n}^2$. We set $\overline{n}$ as 120 throughout



(a) Percentage of jobs meeting deadlines (b) VM utilization ratio (c) Server link utilization ratio (d) Revenue of the cloud provider

Fig. 5. Comparison of the cloud computing models against the jobs' expected deadline extension ratio.



(a) Percentage of successful jobs and provider's revenue (b) VM and server link utilization ratios

Fig. 4. Impact of the profiling relax index.

all the simulations. The bandwidth request for each VM, b, is subject to the normal distribution of $Norm(\bar{b}, 2\bar{b}^2)$. We vary $\bar{b}$ from 50Mbps to 500Mbps to study the impact of requested bandwidth, and set 250Mbps as the default value. The job running duration, p, follows the normal distribution of $Norm(\bar{p}, 2\bar{p}^2)$. We set $\bar{p}$ as 1 hour throughout all the simulations. The deadline extension ratio, $r = \frac{d}{p}$, also follows the normal distribution of $Norm(\bar{r}, 2\bar{r}^2)$. We vary $\bar{r}$ from 1 to 20 to study its impact, and the default value is set as 10.

Tenants' requests dynamically arrive over time following a Poisson process. Since cloud computing workloads usually expose diurnal pattern [21], [22], we set the expected arriving rate of the first 12 hours as λ (simulating the daytime), while that of the second 12 hours as $\frac{\lambda}{5}$ (simulating the night time). We vary λ from 20/hour to 250/hour to study the impact of the job load, and set the default value as 160/hour, indicating that 160 jobs arrive at the cloud per hour in the first 12 hours while 32 jobs arrive per hour in the second 12 hours.

Unless otherwise specified, when studying the impact of a certain parameter in the next subsections, we set all the others parameters as the default values.

Computation Time: We run the simulation in a server equipped with AMD Opteron 4176 2.4GHz 6-core CPU and 32GB RAM. In all groups of simulations, we find that the DCloud resource allocation algorithm takes less than 1 second to allocate a job, which suggests that DCloud can make online resource allocation for dynamically arriving jobs.

B. Profiling Relax Index

We start with the allocation parameter of profiling relax index γ, which is used for masking the job profiling inaccuracy. We vary γ from 0 to 12% (as suggested in [10], the existing profiling method can limit the prediction error below 12%), and set all the other parameters as the default. Note that though we reserve more resource than requested for each job, we do not add the actually required resource to run the job. Under different settings we compute the four metrics and show in Fig. 4.

We observe that the profiling relax index has only minor impact on all four metrics. For example, when γ increases from 0 to 12%, the percentage of successful jobs drops by 2.8% and the provider's revenue drops by 4%. The drops on the VM and bandwidth utilization are 3.5% and 2% respectively. The fundamental reason is that, although we conservatively reserve more resource for a target job, the resource is released to be used by other jobs when the target job finishes earlier than expectation. In all the following simulations, we set γ =12% (worst case) for the DCloud allocation.

C. Parameters of Resource Requests

We then study the impact of two parameters in resource requests, namely, the jobs' expected deadline extension ratio and the mean bandwidth requested per VM.

1) Deadline Extension Ratio: The larger the expected deadline extension ratio $\bar{r}$ is, the larger room there is for a cloud provider to reshape the request. Though this setting may depend on application types, we vary $\bar{r}$ from 1 to 20 to understand its general impact. From Fig. 5, as expected, we can see that the deadline extension ratio has almost no impact on VC, as VC does not consider the job deadline at all. Hence in what follows we focus on discussing BL and DCloud.

In Fig. 5(a), both BL and DCloud complete more jobs with larger $\bar{r}$. For DCloud, the gain comes from larger room for both time sliding and bandwidth scaling. For BL, it is due to the higher probability that a job can finish before a longer deadline. When $\bar{r} = 1$, there is no room for either time sliding or bandwidth scaling, so DCloud performs the same as VC. BL performs the worst among the three, because there is no bandwidth guarantee and some accepted jobs cannot finish before the deadlines. VC can provide bandwidth guarantee and thus the allocated jobs can finish within the deadlines, but its accepted jobs are much fewer than DCloud. When $\overline{r}$ is as high as 20, DCloud can successfully finish about 40% and 80% more jobs than VC and BL, respectively.

A good allocation algorithm should make efficient utilization of VM slots. We show the results in Fig. 5(b). For this metric, VC and BL are not affected by the deadline extension ratio. BL has the highest VM utilization (close to 100%), because it accepts jobs as long as there are any available VM slots, regardless of the bandwidth demands. DCloud performs better than VC, because more jobs can be accepted. The VM utilization of DCloud also grows with higher $\overline{r}$, because the more bandwidth scaling prolongs the VM occupation. When the deadline extension ratio is 20, DCloud's VM utilization is almost three times that of VC.

The server link utilization indicates how well the network is utilized. In Fig. 5(c), all three models do not result in high utilization of server links, due to the oversubscription in higher-level links. VC and BL have similar results, both less than DCloud. It is because the VC results in bandwidth fragmentation, while the competition based bandwidth sharing in BL may cause bandwidth waste if the numbers of flows in server links are imbalanced. We also consider using the total amount of traffic transferred among VMs as another metric. Due to the limited space, we do not include the graphs. DCloud performs even better under this metric. This is because DCloud can make more localized VM allocation by bandwidth scaling as explained in Section III-A, and thus more traffic are transmitted among VMs located in the same physical server.

Evaluating using the profit metric, Fig. 5(d) shows that DCloud significantly outperforms VC and BL, despite that the per job price in DCloud is smaller. The revenue of BL is the lowest though it has the highest VM utilization. When $\overline{r} = 20$, DCloud's revenue is 76.7% and 1,156% higher than that of VC and BL, respectively.

2) Requested Bandwidth per VM: Another important parameter in the resource request is the amount of bandwidth consumed per VM. It reflects the balance between the VM resources and network resources. Intuitively, DCloud has larger advantage with larger bandwidth demand, because we consider balancing the usage of the two resources by bandwidth scaling. Fig. 6 shows the comparison on four metrics with the varied bandwidth.

With a larger bandwidth demand, each VM consumes more resource and thus the percentage of successful jobs will be smaller, as is confirmed in Fig. 6(a). For DCloud and VC, it is more difficult to allocate VMs for jobs with higher bandwidth requests. For BL, higher bandwidth requests also mean longer job running durations because of bandwidth competition, so the probability for a job to violate the deadline is also higher.

With per VM consuming more bandwidth, the overall VM utilization becomes smaller in DCloud and VC, shown in Fig. 6(b), because less jobs can be accepted. However, for DCloud, we find that it is not linearly decreasing. When the bandwidth goes beyond 450Mbps, the VM utilization in DCloud even increases, because of larger room for bandwidth scaling and prolonged VM occupancy.

Similarly, DCloud also outperforms the other two, when evaluating using the server link utilization in Fig. 6(c). The server link utilization in DCloud slowly decreases with higher requested bandwidth, because higher bandwidth per VM indicates more difficulty in scaling the bandwidth to meet the job deadline.

The cloud provider's revenue is shown in Fig. 6(d). When the bandwidth requirement is as low as 50Mbps, DCloud can earn 63.8% and 926% more than VC and BL, respectively. At the other extreme when the bandwidth requirement is as high as 500Mbps, the gap is even more obvious, *i.e.,* DCloud earns 87.0% and 2,170% more than VC and BL, respectively.

D. Network Oversubscription Ratio

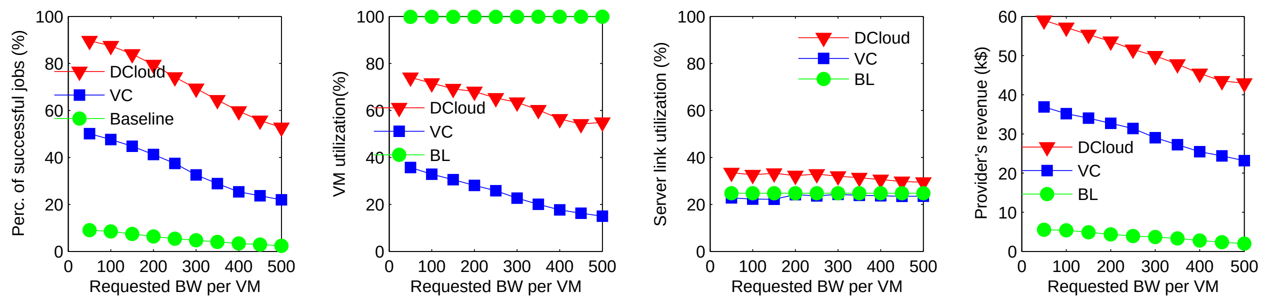
The network oversubscription ratio is an important parameter in data center topology. The larger value it is, the more stringent resource there is in the higher level of the network, which will impact the allocation results. In Fig. 7 we vary the oversubscription ratio from 1:1 to 16:1. From Fig. 7(a), the percentage of successful jobs decreases with higher oversubscription ratio, as less bandwidth is available. DCloud consistently outperforms VC and BL. When the oversubscription ratio is 1:1, the percentages of jobs that can meet their deadlines are 99.1%, 80.9% and 64.3%, in DCloud, VC and BL respectively. When the oversubscription ratio is 16:1, 39.6% jobs can still meet their deadlines in DCloud, 20.7% jobs meet deadlines in VC, while BL can barely finish any jobs within their deadlines.

When the oversubscription ratio increases, higher-level links will be more congested, which in turn affects the job duration and VM occupation. According to Fig. 7(b) we find that when the oversubscription ratio is 1:1, both DCloud and BL can accept all the jobs, and their VM utilizations are both 73.3%, higher than that in VC, 54.8%. When the ratio becomes larger, in BL, the job running duration will be much longer so the VM utilization always reaches 100%. For DCloud, the VM utilization at first grows to 78.6% when the oversubscription ratio is 2:1, because of bandwidth scaling and prolonged VM running duration; but then decreases with even higher oversubscription, because the VM fragments become more difficult to be used if the upper-layer link bandwidth is less. For VC, the VM utilization consistently decreases with higher oversubscription, and is always lower than DCloud.

Fig. 7(c) tells that when there is no oversubscription, DCloud, BL and VC can utilize 73.3%, 68.6% and 55.2% of the server link bandwidth, respectively. Higher oversubscription maps to more congested high-level links, and thus lower server link utilization.

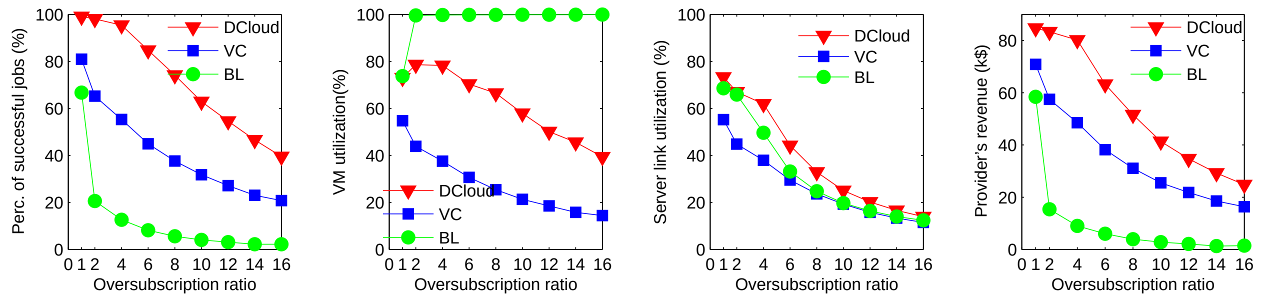
Similarly, the cloud provider's revenue also decreases with the higher oversubscription, illustrated in Fig. 7(d). When there is no oversubscription, DCloud's revenue is 19.7% and 49.1% higher than VC and BL, respectively. With the growth of oversubscription ratio, the gap becomes larger. When the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2489646, IEEE Transactions on Parallel and Distributed Systems



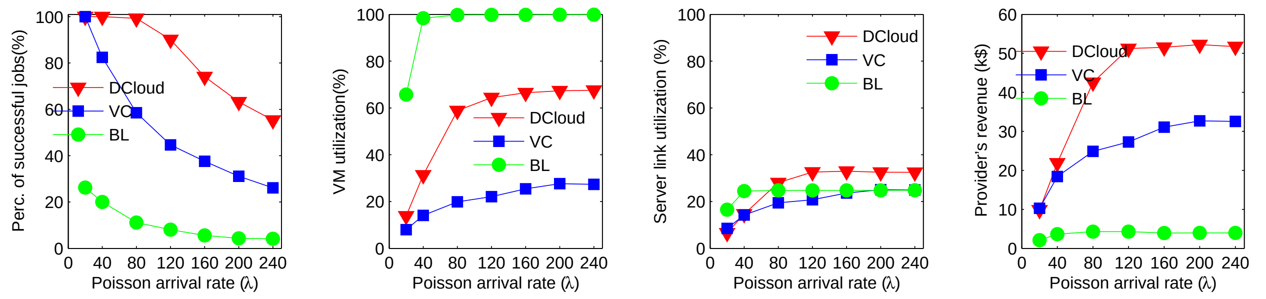
(a) Percentage of jobs meeting deadlines (b) VM utilization ratio (c) Server link utilization ratio (d) Revenue of the cloud provider

Fig. 6. Comparison of the cloud computing models against the mean requested bandwidth per VM.



(a) Percentage of jobs meeting deadlines (b) VM utilization ratio (c) Server link utilization ratio (d) Revenue of the cloud provider

Fig. 7. Comparison of the cloud computing models against the network oversubscription ratio.



(a) Percentage of jobs meeting deadlines (b) VM utilization ratio (c) Server link utilization ratio (d) Revenue of the cloud provider

Fig. 8. Comparison of the cloud computing models against the job arrival rate.

oversubscription ratio is 16:1, DCloud can profit 52.2% and 2,012% more than VC and BL, respectively.

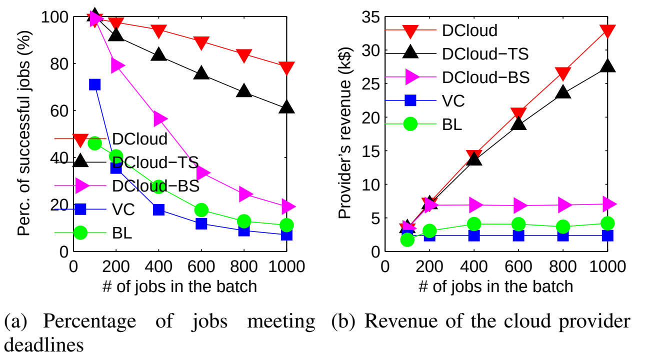
E. Job Arrival Models

The allocation method should be able to handle any form of job arrivals, in order to maintain high profit. We discuss the job arrival process in two ways.

1) Job Arrival Rate in Poisson Process: We first vary the Poisson arrival rate λ of the first 12 hours from 20/hour to 250/hour (correspondingly the Poisson arrival rate of the second 12 hours varies from 4/hour to 50/hour). Fig. 8(a) shows that the percentages of successful jobs in all three models decrease with higher job arrival rate, which is intuitive as the load increases. For example, when $\lambda = 80$/hour, DCloud can still accept almost all the jobs, but VC rejects 40% of the jobs, while BL violates the deadline requirements for about 90% jobs.

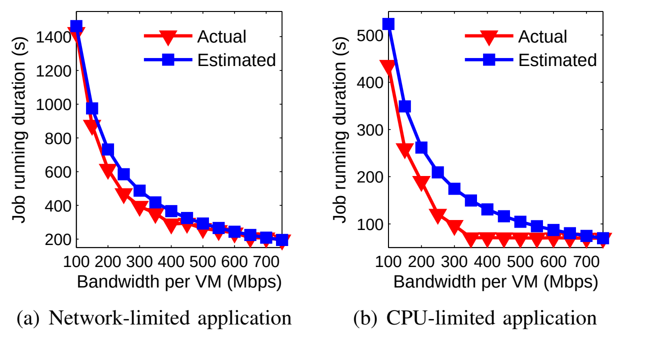
The average VM utilization is shown by Fig. 8(b), which increases with higher job arrival rate. Consistent with previous simulations, BL makes the best utilization of VM resource but many occupied VM slots are wasted without performance guarantee. DCloud outperforms VC by utilizing about 40% more VM resources when $\lambda \geq 80$/hour. The average server link utilization also increases with higher job arrival rate, as is shown in Fig. 8(c). When $\lambda \geq 80$/hour, DCloud can utilize more server link resource than VC and BL allocation.

Fig. 8(d) is the result for cloud provider's revenue. DCloud



(a) Percentage of jobs meeting deadlines (b) Revenue of the cloud provider

Fig. 9. Comparison of the cloud computing models for batch processing.



(a) Network-limited application (b) CPU-limited application

Fig. 10. Estimated and actual job running duration in bandwidth scaling.

brings the highest revenue among the three. The revenues of both DCloud and VC increase with the job arrival rate, since more jobs can be accepted. But for BL, in some cases the revenue even decreases with higher λ. It is because without performance guarantee, accepting more jobs may reversely reduce the number of jobs that can finish within deadlines.

2) Batch Processing: Another possible model of job arrival is the batch processing, in which all the jobs simultaneously arrive at $t = 0$. We also study DCloud resource allocation by time sliding only (we call DCloud-TS) and by bandwidth scaling only (we call DCloud-BS), to understand where the benefit comes from. The results are shown in Fig. 9.

Fig. 9(a) shows the percentage of jobs meeting deadlines against the number of jobs in the batch. DCloud can accept significantly more jobs than VC and BL in batch processing, especially when the number of jobs in the batch is large. Besides, both time sliding and bandwidth scaling help accept more jobs in DCloud.

The revenue of the cloud provider is illustrated in Fig. 9(b). The revenue of DCloud increases with more jobs in the batch, because it can use time sliding and bandwidth scaling to accept more jobs within deadlines. But VC and BL allocation cannot earn more profit even when there are more jobs (>200), because the instantaneous cloud resource is used up. In the batch processing, time sliding in DCloud allocation contributes more than bandwidth scaling in smoothing the peak demand. But by bandwidth scaling only, DCloud can also earn 176% more than VC, and 68.2% more than BL.

F. Summary of the Simulation Results

We summarize our key observations from the simulations as follows. First, the effectiveness of DCloud is not very sensitive to the selection of the profiling relax index. Second, Although BL has the highest VM utilization, the jobs are not guaranteed to finish before deadlines. DCloud has much higher VM utilization than VC and has the highest network utilization among the three, in all scenarios. Third, compared with VC, in most scenarios DCloud can complete more than twice jobs and earns more than 50% for the cloud provider, even with less costs for individual jobs. The gap between DCloud and BL is much more significant. Finally, from analyzing different parameters, we conclude that DCloud has better performance under the settings of larger deadline extension ratio, smaller requested bandwidth per VM, smaller network oversubscription ratio, and higher job arrival rate.

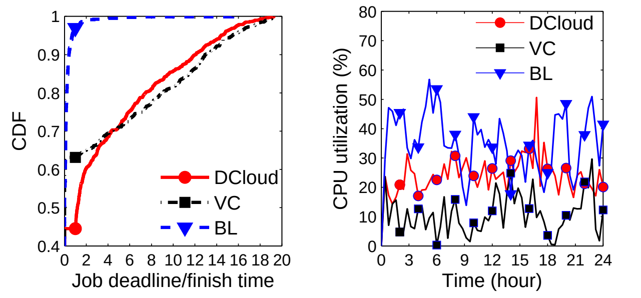
V. Testbed Experiments

Testbed: To validate our assumptions and demonstrate the usefulness in reality, we implement DCloud in a testbed consisting of 16 servers and running real MapReduce Applications. Each server is equipped with a 2-core Intel(R) i3 3.3GHz processor and 4GB memory. They are connected in a three-layer tree structure, by H3C-S5000 switches. The capacity of every link is 1Gbps and the network oversubscription ratio is 4:1. Each server is installed with 2 VMs. We use Linux traffic control API *TC* to provide bandwidth guarantee for each VM. **Sort** and **MapReduce** are chosen as the representative bandwidth-limited and CPU-limited MapReduce jobs respectively.

Using this lab testbed, we first validate the accuracy of estimating the job running duration when scaling the requested bandwidth, and then we emulate request arrival and show the benefit of DCloud.

Bandwidth Scaling: We first run two types of applications in our testbed and measure the job running duration under different VM bandwidths. One application is bandwidth-limited and the other is CPU-limited. Fig. 10 shows the estimated and the actual measured job running durations for the two applications. The original requested bandwidth is 750Mbps. The results demonstrate that when scaling the requested bandwidth, the actual job running duration is always less than but close to our conservative estimation, for both network-limited application and CPU-limited application. It is worth noting that even in network-limited application, the actual job running duration is slightly less than the estimated value. Because in real applications, the operation of sending traffic also occupies some CPU cycles, which leads to conservatively estimated job running duration. The experiment result supports our assumption in reshaping the resource request and our method to adjust bandwidth.

Dynamically Arriving Jobs: Next, we emulate a 24-hour duration of request arrivals, using a Poisson process with



(a) CDF of jobs for the ratio of deadline over finish time (b) Average CPU utilization over time

Fig. 11. Results of the 24-hour experiment.

arrival rate of $\lambda = 10$/hour. The mean job size is 5 VM and the mean requested bandwidth per VM is 500Mbps. The mean job running duration is 1 hour and the expected deadline extension ratio is 10. All the jobs are network-limited.

Again, we compare the allocation algorithms of DCloud, VC and BL. Fig. 11(a) shows the CDF of the ratio of job deadline over actual job finish time. If the ratio is greater than or equal to 1, it means the job is completed within deadline. Otherwise, the job does not finish before deadline. We set the ratio as 0 if the job is not accepted by DCloud or VC. We find that DCloud, VC and BL finish about 55%, 37% and 4% jobs within deadlines, respectively. The cross of the DCloud curve and VC curve implies that VC allocation can finish many jobs more quickly than DCloud because the time sliding and bandwidth scaling used in DCloud will defer the job finish time, but DCloud can accept more jobs by smoothing the peak demand and high-bandwidth requests.

Fig. 11(b) shows the average CPU utilization of all the servers over time. Consistent with the simulation results, BL makes the highest utilization of CPU resource since it accepts more jobs. DCloud makes much higher CPU utilization than VC, because more jobs are accepted. We also evaluate the bandwidth utilization of the three models, and they have similar performance since network is the bottleneck resource in our experiment setting.

VI. Discussions

We have proven many promising features of DCloud. In this section, we discuss its limitations and possible solutions. One major concern of DCloud is its usability, in particular, how for a tenant to profile application to obtain the referenced job running duration, and its impact on resource allocation.

User-facing Web Services: DCloud designed in this paper works well for batch-like computational jobs. But for a user-facing web service, the tenant cannot specify the job deadline or job running duration in the request tuple. DCloud can gracefully encompass such an application, by regarding that the job deadline equals to the job running duration in the request, i.e., $p = d$. In resource allocation, there is no room for time sliding or bandwidth scaling. The charging function becomes $C = (\alpha \times b + \beta) \times n \times p''$, where p'' is the actual running duration of the application in the cloud.

Profiling Cost: Previous work suggests profiling applications by the cloud provider [10], which avoids the profiling cost on tenants. However, in DCloud we do not assume the mutual trust between the cloud provider and tenants, thus the profiling results of the cloud provider are not trusted, as provider might cheat about the results for higher revenue. Therefore, in this paper we still rely on tenants to provide the profiling results. In future, we envision that this task can be done by a third party, which is trusted by both the tenants and provider.

Bad Profiling: DCloud uses the profiling relax index to tolerate underestimated job running duration in a resource request. Actually it is a supererogatory service provided by the cloud provider, since speculative tenants may exploit this room to intentionally submit a job running duration shorter than the actual profiled result (expecting that the cloud provider can prolong it by relaxing). In practice there could still be job requests in which the profiling error exceeds the cloud provider's profiling relax index. Cloud provider has two possible approaches to deal with this kind of jobs. First, the jobs are simply killed at the expected ending times. Second, the cloud provider uses a small portion of the cloud resource to particularly serve these jobs. The VMs of these jobs are immediately migrated to the particular servers at their expected ending times, and then run on a best-effort basis. We will leave a thorough solution to this problem as future work.

VII. Related Work

Recently there are many efforts in the community to address the bandwidth sharing problem in data center networks. In what follows we only discuss the ones closely related with our work.

Bandwidth Competition: Since the current flow based bandwidth competition mechanism based on TCP cannot provide predictable network performance to tenants, VM-level or tenant-level competition mechanisms are proposed, by assigning different weights to VMs or tenants/services. Competition based bandwidth sharing is work-conserving and can make efficient utilization of the network resource, but the network performance of a tenant depends on the number and weights of other tenants in the network. Hence the bandwidth to a tenant is not guaranteed.

Typical solutions include Seawall [24] and Netshare [25]. Seawall [24] designs VM(sender)-level congestion control to replace flow-level congestion control in TCP. Netshare [25] proposes a hierarchical weighted max-min fair sharing mechanism, in which the network manager allocates weights to services and then each service allocates the bandwidth equally to TCP flows.

Hard Bandwidth Guarantee: In order to provide hard bandwidth guarantee and thus predictable application performance, solutions based on static bandwidth allocation and reservation are proposed. They abstract a bandwidth request model for a tenant, assign VMs into proper locations of the network to meet the bandwidth requirement, and use rate limiters to enforce the bandwidth reservation and isolation.

Representative solutions include SecondNet [5], Oktopus [6], TIVC [8] and [9]. SecondNet [5] uses a bandwidth matrix to describe the tenant's bandwidth request, and designs heuristic algorithms to allocate VMs based on both network bandwidth and VM resource. Instead, Oktopus [6] uses a hose model to abstract the tenant's bandwidth request, including both *virtual clusters* and *oversubscribed virtual clusters*, and proposes VM allocation algorithm for homogeneous bandwidth request. TIVC [8] follows the basic idea of Oktopus, but improves the bandwidth utilization efficiency by capturing the time-interleaving dynamical bandwidth requirement from a tenant. In [9], authors focus on heterogeneous bandwidth demand between VMs, and design a heuristic allocation algorithm. Bazaar [10] proposes flexibly varying the number of VMs and bandwidth to better utilize cloud resource with application performance guarantee, with a focus on MapReduce computation.

Bandwidth Competition with Minimum Bandwidth Guarantee: Hard bandwidth guarantee usually results in inefficient utilization of network resource. In order to be work-conserving, bandwidth sharing mechanisms based on competition with minimum guarantee are proposed recently. The basic idea is to guarantee a minimum share of bandwidth for tenant networks, while letting them compete for the residual bandwidth.

Related solutions include FairCloud [11], ElasticSwitch [12] and EyeQ [13]. Faircloud analyzes the fundamental tradeoff in today's weight-based bandwidth sharing solutions, and the proposed PSP mechanism provides bandwidth guarantees and is work-conserving, which requires updating the switch hardware. Instead, ElasticSwitch depends on a bandwidth negotiation scheme inside hypervisors to realize efficient bandwidth sharing with minimum guarantee, and is thus more implementable. EyeQ also designs a programmable bandwidth arbitration mechanism to efficiently utilize the network resource with minimum guarantee to tenants, but it requires a congestion-free core network.

VIII. Conclusion

In this paper we designed DCloud, a resource allocation approach for cloud computing jobs to both meet their deadlines and efficiently utilize the cloud resource. By requiring a tenant to submit both her job deadline and the resource demand, DCloud provides predictable performance to applications and also leaves room for shaping the resource requests to better match the residual resource. DCloud uses time sliding and bandwidth scaling to determine the most appropriate time interval to launch each job, as well as the VM locations and reserved link bandwidth. A charging mechanism to encourage selfish tenants to submit the actual required resource, which makes the resource allocation algorithms work more effectively. Extensive simulations and testbed experiments show that, compared with the baseline allocation and recently proposed VC allocation, DCloud can finish significantly more jobs within deadlines, make better utilization of the VM and network resource, and gain much more revenue.

References

[1] "Amazon Elastic Compute Cloud." www.aws.amazon.com/ec2.
[2] "Google App Engine." www.code.google.com/appengine/.
[3] "Windows Azure Platform." www.microsoft.com/windowsazure/.
[4] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks.," in *WIOV*, 2011.
[5] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of CoNext'10*.
[6] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of SIGCOMM'11*.
[7] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. Ramakrishnan, and J. E. Van der Merwe, "Resource management with hoses: point-to-cloud services for virtual private networks," *IEEE/ACM Transactions on Networking,*, vol. 10, no. 5, pp. 679–692, 2002.
[8] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *Proceedings of SIGCOMM'12*.
[9] J. Zhu, D. Li, J. Wu, H. Liu, Y. Zhang, and J. Zhang, "Towards bandwidth guarantee in multi-tenancy cloud computing networks," in *Proceedings of ICNP'12*.
[10] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the tenant-provider gap in cloud services," in *Proceedings of SoCC'12*.
[11] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *Proceedings of SIGCOMM'12*.
[12] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing," in *Proceedings of SIGCOMM'13*.
[13] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "Eyeq: Practical network performance isolation at the edge," in *Proceedings of NSDI'13*.
[14] "Amazon EMR." http://aws.amazon.com/cn/elasticmapreduce/?hp=tile.
[15] "Hadoop at Rackspace." https://developer.rackspace.com/databases/hadoop/.
[16] N. Joukov, A. Traeger, R. Iyer, C. Wright, and E. Zadok, "Operating system profiling via latency analysis," in *Proceedings of OSDI'06*.
[17] B. Cantrill, M. Shapiro, and A. Leventhal, "Dynamic instrumentation of production systems," in *Proceedings of USENIX ATC'04*.
[18] G. Wang, A. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in mapreduce setups," in *Proccedings of MASCOTS'09*.
[19] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of OSDI'04*.
[20] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of SIGCOMM'09*.
[21] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Dynamic provisioning of multi-tier internet applications," in *Proceedings of ICAC'05*.
[22] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Proceedings of IEEE NOMS'12*.
[23] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks.," in *Proceedings of NSDI'10*.
[24] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proceedings of NSDI'11*.
[25] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "NetShare: Virtualizing Data Center Networks across Services," in *Technical Report, UCSD, 2010*.