Technical Report: CS-TR-2017-005
Virtual Machine Provisioning for Applications with Multiple Deadlines in Resource-Constrained Clouds (Extended Version)

Rehana Begam, Wei Wang and Dakai Zhu
University of Texas at San Antonio
San Antonio, Texas 78249
Email: {ylm213@my.utsa.edu; wei.wang@utsa.edu; dakai.zhu@utsa.edu}

***Abstract*—Recently, several studies have considered applications with a single time constraint (i.e., deadline) running on cloud systems. In this work, to effectively support user requests with flexible timing constraints (e.g., users may prefer expedited services and are willing to pay extra for getting their job processed at earlier times), we consider applications with multiple deadlines for being processed in resource-constrained cloud systems and investigate corresponding virtual machine (VM) provisioning schemes. Specifically, by considering the multiple deadline-bid pairs of user requests, we propose a *Slope-based* Time-Sensitive Resource Factor with Dominant Resource being considered to prioritize such requests. In addition, we study the mapping schemes that allocate the multiple VMs of a user request to only one or multiple computing nodes, which are denoted as *Bundled* vs. *Distributed* mappings, respectively. The evaluation results show that, compared to the single deadline schemes, the proposed VM provisioning schemes with multiple deadlines and distributed mapping can significantly improve the overall resource utilization and system benefit.**

1. Introduction

In cloud computing, resource allocation is the important process of assigning available resources to the needed cloud applications over the internet and service requests may starve if it is not done properly [1]. The computing resources in cloud systems are normally provided to serve requests in the form of virtual machines (VMs). The problem of VM provisioning in clouds has been investigated from different points of view and many research studies have been reported on resource allocation for cloud systems [2], [3], [4]. For instance, the VM provisioning techniques have been proposed to improve user application performance [5], [6], [7], [8], to efficiently utilize cloud resources [9], [10], [11], [12], to minimize the user cost and maximize the revenue for cloud service providers [13], [14], [15], [16], [17], or to deliver services even in presence of failures [18], [19], [20], [21].

However, only limited research has focused on applications with timing constraints (i.e., deadlines) [7], [22], [23], [24], [25]. Note that, many applications running on cloud systems do have various timing-constraints. For instance, 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. [26], [27], [28]) and most of these jobs have timing constraints, where the results may become useless if they cannot be processed in a certain time. Moreover deadlines can also be considered as a performance assurance metric in the Service Level Agreement (SLA) [22].

Mao and Humphrey employed integer programming to determine the optimal auto-scaling policies for jobs with deadlines and budget constraints on public clouds [29], [30]. A deadline-driven resource manager for scientific applications running on hybrid clouds was studied in [25]. Le et al. compared the performance of several classic scheduling algorithms for jobs with deadlines in cloud systems, including first-come-first-serve, shortest job first and EDF algorithms [7]. Zhu et al. investigated a real-time workflow fault-tolerant model and a dynamic fault-tolerant scheduling algorithm for real-time scientific workflows in virtualized clouds [31]. In [23], Li et al. introduced the DCloud resource allocator that leverages the (soft) deadlines of jobs in public clouds. By *time sliding* (delaying the launching of a job) and *bandwidth scaling* (dynamic reduction of network bandwidth), they try to match the resources allocated to jobs with the clouds residual resources.

In our recent work [32], by focusing on resource-constrained clouds, we have studied time-sensitive VM provisioning schemes for applications with a single deadline. Specifically, by considering the Time-Sensitive Resource Factor (TSRF) and Dominant Resource (DR) of user requests, we proposed two prioritization schemes and Euclidean-Distance based mapping approach. However, in these existing studies, they usually consider user requests with only one timing constraint (i.e., deadline).

In order to support flexible timing requirements of user requests, it can be beneficial to allow user requests to have multiple timing-constraints (i.e., deadlines). For instance, some users may prefer expedited services with extra cost to get their jobs processed at earlier times. For cases where there are bursty user requests and not all of them can be processed in time, some user requests may accept degraded services as long as their applications can be processed by some extended deadlines with much lower cost.

In this work, we consider such user requests that have multiple deadlines to specify their flexible service and cost requirements. By focusing on resource-constrained (such as private) cloud systems, we investigate efficient VM provisioning schemes with the objective of improving resource utilization, processing more user requests in time and achieving better system benefits.

In particular, based on the slope of user request's bid vs. deadline (i.e., how bid decreases when deadline increases), we propose the slope-based the Time-Sensitive Resource Factor (TSRF) with Dominant Resource (DR) to prioritize the requests. Then, by allowing the multiple demanded VMs of a user request be allocated to only one or different computing nodes in the cloud, we study both *Bundled* and *Distributed* mapping schemes. The evaluation results show than, compared to the schemes that consider only a single deadline, the proposed multi-deadline based schemes can lead to much better resource utilization and system benefit.

The remainder of this paper is organized as follows. Section 2 presents the system models. The multi-deadline based time-sensitive VM provisioning schemes are proposed in Section 3 followed by a motivational example. The evaluation results are discussed in Section 4 and Section 5 concludes the paper.

2. System Models

2.1. Resource-Constrained Cloud

The same as in our recent work [32], we consider a resource-constrained (private) cloud system that consists of M computing nodes $\Pi = \{\mathbb{N}_1, \ldots, \mathbb{N}_M\}$. There are R types of resources $\Gamma = \{\mathbb{R}_1, \ldots, \mathbb{R}_R\}$ in the system (such as CPU cores and memory). Each node $\mathbb{N}_k$ represents a pool of such resources with its capacity vector being denoted as $\mathbb{N}_k = (c_k^1, \ldots, c_k^R)^T$. Here, c_k^r represents the total capacity of resource $\mathbb{R}_r$ on node $\mathbb{N}_k$ (e.g., number of CPU cores). With heterogeneous nodes being considered, the capacity of the one resource in different nodes can be different.

The computing resources in the cloud can be accessed by cloud users in the form of virtual machines (VMs). In this work, we consider V types of virtual machines that have different resource requirements. For the VM type $\mathbb{V}_k$, its required resources can be denoted by a demand vector $\mathbb{V}_k = (w_k^1, \ldots, w_k^R)^T$, where w_k^r represents the required capacity (or amount) of resource $\mathbb{R}_r$.

As an example, focusing on two types of resources (CPU and memory), we can consider a cloud system with two nodes $\mathbb{N}_1$ and $\mathbb{N}_2$. Suppose that there are 16 CPU cores and 128 GB memory in node $\mathbb{N}_1$ and 4 CPU cores and 8 GB memory in node $\mathbb{N}_2$. Suppose that there are four types of virtual machines with their resource demand vectors as $\mathbb{V}_1 = (1, 2)^T$, $\mathbb{V}_2 = (2, 8)^T$, $\mathbb{V}_3 = (2, 16)^T$ and $\mathbb{V}_4 = (4, 32)^T$ (where the numbers mimic the ones for Amazon EC2 VM instances `t2.small`, `t4.large`, `r4.large`, and `t4.xlarge` [33]), respectively.

To guarantee the performance of requested VMs to cloud users, similar to other recent works [34], [35], [36], we assume that there is no sharing of resources among the VMs that are simultaneously mapped to the same node. That is, a physical CPU core from a node will be allocated for each virtual CPU demanded by a VM, and the same for other resources. Moreover, for simplicity, we consider only the capacity/demand of a resource (e.g., number of cores or amount of memory) in a node/VM without differentiating its actual model and specification (such as core frequency or memory speed). Exploring those features is beyond the scope of this paper and will be addressed in our future work. In addition, we further assume that a VM can only be mapped to a single node where all its demanded resources have to be supported by that node [34], [35], [36].

2.2. User Requests with Multiple Deadlines

Similar to our recent work [32], in addition to the application to be processed, the request for the demanded virtual machines can be represented as a tuple $\theta_i = (a_i, \langle v_i, m_i \rangle, t_i, \mathbf{DB}_i)$. Here, a_i denotes the arrival time of the user request. To run the desired application, we assume that the user knows the type v_i ($1 \leq v_i \leq V$) and number m_i of VMs needed as well as the (worst case) operation time t_i to execute the application, which includes the VMs' setup and tear down overheads.

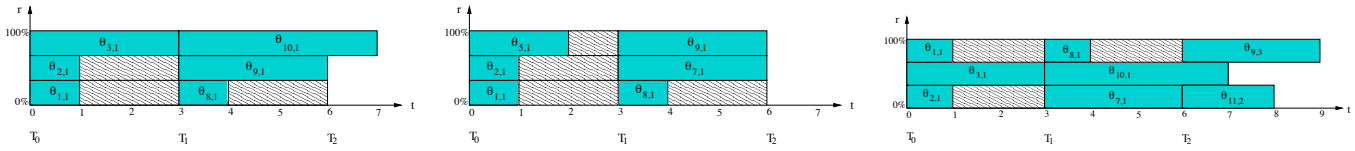
To model the flexible timing requirements of user requests, the vector $\mathbf{DB}_i$ specifies the list of service levels where the level j is represented by a pair of deadline and bid $\langle d_{i,j}, b_{i,j} \rangle$. Here, $d_{i,j}$ is the j'th deadline of request θ_i and $b_{i,j}$ represents the corresponding bid for the application being processed by $d_{i,j}$. We use bid as a generic term to represent the user cost or system benefit. We assume that there are l_i service levels for request θ_i. That is, we have $\mathbf{DB}_i = \langle \langle d_{i,1}, b_{i,1} \rangle, \ldots, \langle d_{i,j}, b_{i,j} \rangle, \ldots, \langle d_{i,l_i}, b_{i,l_i} \rangle \rangle$. Without loss of generality, we assume that $d_{i,j} < d_{i,j+1}$ and $b_{i,j} > b_{i,j+1}$ where $j \in [1, l_i - 1]$.

As an example, suppose that a user request arrives at time 5, which requires 2 VMs of type $\mathbb{V}_1$ to run its application for 10 time units. Here, if the application can be processed within 20 time units the user specified cost is 15; similarly, if it is served within 25 time units, the cost is 10; and 35 time units with the cost of 2. The user request can be represented as $\theta = (5, \langle v_1, 2 \rangle, 10, \langle \langle 20, 15 \rangle, \langle 25, 10 \rangle, \langle 35, 2 \rangle \rangle)$.

The key notations used in this work is summarized in Table 1.

TABLE 1: Key Notations

Notations	Definitions
$\mathbb{N}_k$	A cloud node
c_k^r	Capacity of resource $\mathbb{R}_r$ of node $\mathbb{N}_k$
$\mathbb{R}_r$	A resource type
$\mathbb{V}_k$	A VM type
w_k^r	Required capacity of resource $\mathbb{R}_r$ for VM type $\mathbb{V}_k$
θ_i	A user request
a_i	Arriving time of user request θ_i
m_i	Requested VM count of user request θ_i
t_i	Execution time (worst case) of user request θ_i
$\mathbf{DB}_i$	vector of all deadline vs. bid pairs for user request θ_i
$d_{i,j}$	The j'th deadline of user request θ_i
$b_{i,j}$	Bid for the deadline d_{i,l_i} of user request θ_i
l_i	Number of deadline vs. bid pairs of user request θ_i



a. EDF b. TSRF c. multi-TSRF

Figure 1: Comparison of Different Schemes

3. VM Provisioning with Multiple Deadlines

3.1. A Motivational Example

To illustrate how multi-deadline clouds can improve overall system utilization and benefit over a single deadline one, let's consider a time-sensitive cloud system with a single node $\mathbb{N}_1$ of two types of resources: CPU and memory. There are 7 CPU cores and 8GB memory in the system. Two types of VMs will be provided, which are $\mathbb{V}_1$ and $\mathbb{V}_2$ with their demand vectors as $\mathbb{V}_1 = (2,2)^T$ and $\mathbb{V}_2 = (3,3)^T$, respectively. All the multi-deadline requests will have three levels of deadline constraints that is $\forall\theta_i \in \Theta, |L_i| = 3$ and each $\theta_{i,j}$ represents the request for user i with the current deadline level j.

Let's assume the length of each interval T is 5 time units. There are six (6) user requests arrived at the start of interval T_0 and five (5) more at (or before) the start of the next interval T_1. For the single deadline time-sensitive system, these requests will have only one deadline and one bid. On the other hand, for the multi-deadline system they will have two additional ⟨deadline, bid⟩ pairs (totaling in three pairs) associated with them. Table 2 shows both the single deadline (column 2) and multi-deadline (column 3) formats for the same user requests.

TABLE 2: Requests to be processed in the intervals

θ_i	single: $(a_i, v_i, t_i, d_i, b_i)$	multiple: $(a_i, \langle v_i, m_i\rangle, t_i, L_i)$
θ_1	$(0,1,1,1,9)$	$(0,\langle 1,1\rangle,1,\langle\langle 1,9\rangle,\langle 3,8\rangle,\langle 4,2\rangle\rangle)$
θ_2	$(0,1,1,1,7)$	$(0,\langle 1,1\rangle,1,\langle\langle 1,7\rangle,\langle 2,6\rangle,\langle 3,2\rangle\rangle)$
θ_3	$(0,1,3,3,6)$	$(0,\langle 1,1\rangle,3,\langle\langle 3,6\rangle,\langle 5,2\rangle,\langle 6,1\rangle\rangle)$
θ_4	$(0,1,4,4,10)$	$(0,\langle 1,1\rangle,4,\langle\langle 4,10\rangle,\langle 6,4\rangle,\langle 8,1\rangle\rangle)$
θ_5	$(0,1,2,3,7)$	$(0,\langle 1,1\rangle,2,\langle\langle 3,7\rangle,\langle 6,2\rangle,\langle 7,1\rangle\rangle)$
θ_6	$(0,1,2,3,6)$	$(0,\langle 1,1\rangle,2,\langle\langle 3,6\rangle,\langle 6,3\rangle,\langle 7,1\rangle\rangle)$
θ_7	$(1,1,3,3,15)$	$(1,\langle 1,1\rangle,3,\langle\langle 3,15\rangle,\langle 4,7\rangle,\langle 5,5\rangle\rangle)$
θ_8	$(1,1,1,1,8)$	$(1,\langle 1,1\rangle,1,\langle\langle 1,8\rangle,\langle 2,6\rangle,\langle 3,3\rangle\rangle)$
θ_9	$(1,1,3,2,9)$	$(1,\langle 1,1\rangle,3,\langle\langle 2,9\rangle,\langle 5,7\rangle,\langle 6,2\rangle\rangle)$
θ_{10}	$(1,1,4,2,10)$	$(1,\langle 1,1\rangle,4,\langle\langle 2,10\rangle,\langle 3,5\rangle,\langle 8,1\rangle\rangle)$
θ_{11}	$(1,1,2,3,6)$	$(1,\langle 1,1\rangle,2,\langle\langle 3,6\rangle,\langle 6,3\rangle,\langle 7,1\rangle\rangle)$

Figure 1 shows the selection and execution of user requests for different single and multiple deadline schemes to explain how multi-deadline system can improve the performance. At the beginning of each interval, these schemes allocate all available resources to the requests based on their priorities until resources are exhausted. Figure 1a shows the allocation for *single-deadline EDF* []. This scheme prioritizes a user request purely based on its only deadline. This scheme can earn 49 bid values by serving 6 users from 11 users. The *single-deadline TSRF* [] scheme considers its only deadline and bid and requests are prioritized based on the TSR factor values. This scheme is shown in Figure 1b. It can earn 55 bid values by serving 6 users from 11 users. At each interval, both of these will discard the missed requests.

Figure 1c shows the allocation of resource management scheme that considers multiple deadlines. This scheme allocates resource for $\theta_{2,1}$, $\theta_{3,1}$ and $\theta_{1,1}$ at interval T_0 and the remaining ones miss their first level deadline. Same goes for interval T_1 as well. Instead of immediately discarding the missed requests, they are given two more chances (as they still have two other deadlines). The next available deadline level that can still be met are considered for each of these remaining requests and are considered again for the next intervals. As a result, the scheme can eventually allocate resources for $\theta_{11,2}$ and $\theta_{9,3}$ considering their extended deadline levels at interval T_2. Thus this scheme achieves 60 bid values and serves 8 requests among 11 which are higher than the previous two.

This example clearly shows that having multiple time constraints for a request provides more flexibility for the resource provisioning mechanism. With an efficient multi-deadline resource provisioning algorithm, cloud systems can serve more user requests with timing constraints and also achieve higher overall reward.

For user requests with a single deadline, we have studied the prioritization heuristics based on the Time-Sensitive Resource Factor (TSRF) and Dominant Resource (DR) [32]. Note that, even if there is only one deadline vs. bid pair as in [32], finding the optimal VM provisioning for user requests is a general bin packing problem and is NP-hard. Therefore, in this work, by extending the idea of Time-Sensitive Resource Factor (TSRF) and considering the multiple deadlines of user requests, we focus on investigating efficient prioritization and mapping heuristics.

3.2. Overview of VM Provisioning

Similar to [32], we consider an interval-based strategy for VM provisioning and Algorithm 1 shows the main steps at the beginning of each interval T. It first collects the newly arrived user requests as well as those that have not been served yet to form the request set (line 2). For each computing node, the available capacity of its resources is updated (line 3 with the help of procedure *UpdateAvailCapacity()*).

For the two main steps: requests prioritization and allocation of requests' VMs (line 5 and line 8 respectively), we will discuss them in details in the next two sections.

Algorithm 1 : VM Provisioning at each interval T

1: //collect user requests and available resource capacities
2: $\Theta(t_0) = \{\theta_i | \theta_i \in$ (Arrival or Wait Queues)$\}$;
3: $\Pi =$ *UpdateAvailCapacity()*;
4: **for** $(\theta_i \in \Theta(t_0)$) **do**
5: $\mathbf{Q} \leftarrow$ *Prioritize*(θ_i, $*$); //$*$ is M-TSRF or M-TSRF/DR
6: **end for**
7: //allocate requested VMs to computing nodes
8: *AllocateVMs*($\mathbf{Q}$, Π);
9:
10: **Procedure** Prioritize(θ_i, *)
11: // j is current deadline level of θ_i
12: *calculate* $sl_{i,j}$; [eqn(1)]
13: *calculate* $\xi_{i,j}$ or $\tau_{i,j}$; [eqn(4) or eqn(6)]

3.3. Slope-based Multi-Deadline Prioritization

Note that, as discussed in Section 2, an important feature for the user requests with multiple pairs of deadlines and bids is that, the deadline and its associated bid are correlated. In general, bid decreases as deadline increase and vice versa. To obtain the maximum system benefits (i.e., overall achieved bids), we should exploit such correlation of user requests. Intuitively, when deadline increases, the user request with the largest drop of its bid should be served at an earlier times. Following this idea, we propose the *slope-based* approach regarding to user requests' bids vs. deadlines when prioritizing them.

As an example, considering four (4) user requests θ_1, θ_2, θ_3 and θ_4, with their service vectors as $\mathbf{DB}_1 = \langle\langle 1, 9\rangle, \langle 3, 8\rangle, \langle 4, 2\rangle\rangle$, $\mathbf{DB}_2 = \langle\langle 1, 7\rangle, \langle 2, 6\rangle, \langle 3, 2\rangle\rangle$, $\mathbf{DB}_3 = \langle\langle 2, 7\rangle, \langle 5, 2\rangle, \langle 6, 1\rangle\rangle$ and $\mathbf{DB}_4 = \langle\langle 3, 10\rangle, \langle 3, 4\rangle, \langle 9, 1\rangle\rangle$.

Figure 2 shows the bid vs. deadline relation for these four user requests. Here, we can see that for user request θ_1, the line segment from point $\langle 3, 8\rangle$ to point $\langle 4, 2\rangle$ is steeper than the line segment from point $\langle 1, 9\rangle$ to point $\langle 3, 8\rangle$. It means that the bid in the second pair would drop faster to the next level than the one in the first pair, and we should try to process request θ_1 by its second deadline 3. Moreover, the bid in the second pair of request θ_1 also drops (from $\langle 3, 8\rangle$ to $\langle 4, 2\rangle$) faster than that of the second pair of request θ_2 (from $\langle 2, 6\rangle$ to $\langle 3, 2\rangle$). It means that the request θ_1 should have higher priority with its second deadline vs. bid pair being considered when compared to that of the request θ_2.

From their bid-deadline curves, we define the slope for the j'th level of a user request θ_i as:

$$sl_{i,j} = \frac{\|b_{i,j} - b_{i,(j-1)}\|}{\|d_{i,j} - d_{i,(j-1)}\|} \tag{1}$$

Then, considering the multiple deadline of request θ_i, the *Modified Time-Sensitive Resource Factor (mTSRF)* $\hat{\eta}^k_{i,j}$ to indicate the resource usage efficiency of θ_i on a computing node $\mathbb{N}_k$ with its j's deadline can be defined as:

$$\hat{\eta}^k_{i,j} = \frac{b_{i,j}}{(d_{i,j} - t_0) \cdot t_i \cdot \sum_{\mathbb{R}_r \in \Gamma} \frac{y^r_i}{c^r_k}} \tag{2}$$

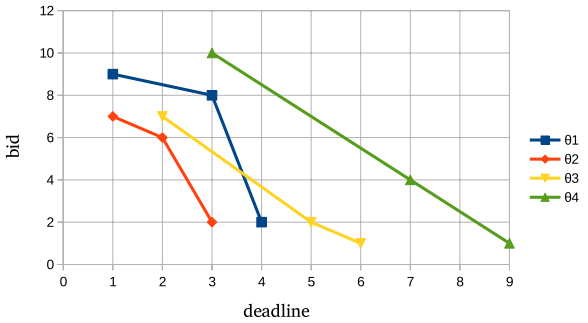


Figure 2: Slope of bid-deadline curves

where $y^r_i = \sum_{\mathbb{R}_r \in \Gamma} m_i \cdot w^r_{v_i}$ denotes the total demand of θ_i for resource $\mathbb{R}_r$; c^r_k is the available resource capacity of resource $\mathbb{R}_r$ in the node $\mathbb{N}_k$. By considering all eligible computing nodes that have enough remaining resources for θ_i, we can formulate *Modified Time-Sensitive Resource Factor (mTSRF)* as $\hat{\eta}_{i,j}$:

$$\hat{\eta}_{i,j} = \max_{\mathbb{N}_p \in \Pi} \{\hat{\eta}^k_{i,j} | \forall \mathbb{R}_r \in \Gamma, y^r_i \leq c^r_k\} \tag{3}$$

Here, with the inversely proportional relation to the overall resource usage ratio, the factor will have a higher value when the required resources are relatively less on one node. Similarly, smaller deadlines, less required time and larger benefit values can also lead to higher values for $\hat{\eta}^k_{i,j}$. In addition to urgency of θ_i, higher values of $\hat{\eta}^k_{i,j}$ also indicate that more system benefit can be achieved with less resources.

Finally, by incorporating the corresponding slope with the *mTSRF*, the *Slope-based Multi-Deadline TSRF* (***S-M-TSRF***) $\xi_{i,j}$ is defined as:

$$\xi_{i,j} = sl_{i,j} \cdot \hat{\eta}_{i,j} \tag{4}$$

Similarly, we can extend θ_i's dominant share ds_i and define its *Modified TSRF with Dominant Resource (mTSRF/DR)* $\hat{\delta}_{i,j}$ and the *Slope-based Multi-Deadline TSRF with Dominant Resource* (***S-M-TSRF/DR***) $\tau_{i,j}$ for the j'th deadline level of θ_i as:

$$\hat{\delta}_{i,j} = \frac{b_{i,j}}{(d_{i,j} - t_0) \cdot t_i \cdot ds_i} \tag{5}$$

$$\tau_{i,j} = sl_{i,j} \cdot \hat{\delta}_{i,j} \tag{6}$$

Once the ***S-M-TSRF*** or ***S-M-TSRF/DR*** factor values for all requests are obtained, we can prioritize the requests accordingly where the ones with larger factor values are assigned higher priorities (lines 10 to 13 in Algorithm 1). When two requests have the same value, tie can be broken arbitrarily.

3.4. Mapping of VMs: Bundled vs. Distributed

An inefficient request-to-node mapping strategy could lead to poor resource utilizations and system performance. In our recent work [32], we proposed an *Euclidean Distance*

(ED) based mapping heuristic for requests requiring only a single VM. When multiple VMs are needed for a user request to run its application, it is possible that the VMs do not fit in any single computing node but be distributed among multiple computing nodes. Considering such cases, we propose two different mapping in this work: ***Bundled*** and ***Distributed***.

3.4.1. Euclidean Distance for Bundled Mapping. For the *Bundled-based* mapping, all the VMs' resource requirements are considered together as the total resource demand of the request when calculating its Euclidean Distance. Let's consider a request θ_i that needs k_i VMs of type $\mathbb{V}_{v_i}$ with its demand vector as $\mathbf{D}_{v_i} = (y_i^1, \ldots, y_i^R)^T$. Then the *Bundled-Euclidean-Distance* $b\text{-}ed_i^k$ of θ_i on node $\mathbb{N}_k$ can be defined as:

$$\text{b-ed}_i^k \quad = \quad \sqrt{\sum_{\mathbb{R}_r \in \Gamma} \left(\frac{c_k^r}{c_k^1} - \frac{y_i^r}{y_i^1} \right)^2} \tag{7}$$

where $y_i^r = \sum_{\mathbb{R}_r \in \Gamma} m_i \cdot w_{v_i}^r$ denotes the total demand of θ_i for resource $\mathbb{R}_r$ and $\mathbb{N}_k = (c_k^1, \ldots, c_k^R)^T$ is the available capacity vector for node $\mathbb{N}_k$ at time t.

Algorithm 2 : Function *ALLOC(*Π*, θ_i, ED)* at interval T

1: $\Pi_i^{eligible} = \emptyset$; $ed_i = \infty$; $p_i = -1$; //initialization
2: //Find all eligible computing nodes for the request θ_i
3: **for** (each $\mathbb{N}_k \in \Pi$) **do**
4: **if** ($\mathbb{N}_k \geq \mathbf{D}_{v_i}$) **then**
5: Add node $\mathbb{N}_k$ to $\Pi_i^{eligible}$;
6: **end if**
7: **end for**
8: **for** (each $\mathbb{N}_k \in \Pi_i^{eligible}$) **do**
9: Calculate ed_i^k; [eqtn (7) or eqtn (8)]
10: **if** ($ed_i^k < ed_i$) **then**
11: $ed_i = ed_i^k$; $p_i = k$; //node $\mathbb{N}_k$ is better
12: **end if**
13: **end for**
14: **Output:** p_i

3.4.2. Euclidean Distance for Distributed Mapping. Unlike the bundled mapping, for the *Distributed* mapping, it considers one VM at a time when calculate the request's Euclidean Distance. Let's consider a request θ_i that needs m_i VMs of type $\mathbb{V}_{v_i}$. For each of these VMs, it will call the mapping Algorithm 2 (m_i times in total) and is considered to be successful only when all the VMs can be served at the same interval T. For this mapping, the demand vector for any one VM would be $\mathbf{D}_{v_i} = (w_{v_i}^1, \ldots, w_{v_i}^R)^T$. And the *Distributed-Euclidean-Distance* $d\text{-}ed_i^k$ of θ_i on node $\mathbb{N}_k$ can be defined as:

$$\text{d-ed}_i^k \quad = \quad \sqrt{\sum_{\mathbb{R}_r \in \Gamma} \left(\frac{c_k^r}{c_k^1} - \frac{w_{v_i}^r}{w_{v_i}^1} \right)^2} \tag{8}$$

where $\mathbb{N}_k = (c_k^1, \ldots, c_k^R)^T$ is the available capacity vector for node $\mathbb{N}_k$.

Algorithm 2 shows the basic steps of our mapping schemes, which can be used with both bundled and distributed Euclidean Distances of a user request. It first finds all eligible nodes $\mathbb{N}_p$ where there is enough resource to serve the θ_i request's resource demand $\mathbf{D}_{v_i}$ (line 3 to 7). It then calculates the euclidean distance for all those eligible nodes and selects the one with the minimum distance (line 8 to 13). For the *Bundled* and *Distributed* approaches, they calculates the total resource demand $\mathbf{D}_{v_i}$ and euclidean distance differently, as discussed above, which results in different mapping of the same request.

Algorithm 3 : VM allocation at interval T

1: $\mathbf{S} \leftarrow \emptyset$; // selected user requests set
2: **while** ($\mathbf{Q}$ is not empty) **do**
3: $\theta_i \leftarrow top(\mathbf{Q})$
4: **if** (*isValid*(θ_i)) **then**
5: p_i = *ALLOC*(θ_i, Π, #); // # is bundled- or distr-ED
6: **if** ($p_i \in \Pi$) **then**
7: $\mathbf{S} \leftarrow \mathbf{S} \cup \langle \theta_i, p_i \rangle$;
8: $\mathbf{Q}.pop()$;
9: *UpdateAvailCapacity()*;
10: **else**
11: $\mathbf{Q}.pop()$; // no suitable p_i in this T
12: Add(θ_i, Wait Queue);
13: **end if**
14: **else**
15: **if** ($j < max\_level$) **then**
16: $j \leftarrow j + 1$
17: $\mathbf{Q} \leftarrow$ *Prioritize(*$\hat{\theta}_i$, $*$*)*;
18: **else**
19: $\mathbf{Q}.pop()$; // discarded
20: **end if**
21: **end if**
22: **end while**
23: **Output: S**

3.5. VM Allocation for Requests with Multiple Deadlines

Finally, the steps for allocating the VMs of user requests with multiple deadlines are summarized in Algorithm 3. For all the user requests from the queue, the algorithm first checks if the current j'th deadline of the top request θ_i can be met (procedure *isValid(*θ_i, j*)* at line 4) or not. If the deadline can be met, then procedure *ALLOC(*θ_i, Π, #*)* finds an appropriate computing node p_i based on the selected allocation mechanism. The capacity of the selected node is then updated and the request is removed from the $\mathbf{Q}$ and put into the selected users set. If no suitable node can be found, the request is put back to the waiting queue.

If the current j'th deadline of the request can not be met, the algorithm looks for the request's next deadline (if any), and updates its parameters accordingly and puts back

the updated request $\hat{\theta}_i$ in the priority queue $\mathbf{Q}$ with a new priority. If the request has already exhausted all its deadlines, it cannot be served by its latest deadline and is discarded.

4. Evaluations and Discussions

TABLE 3: VM configurations and prices

VM	CPUs	Memory(GB)	bid/time unit
c4.large	2	3.75	[0.05, 0.15]
c4.xlarge	4	7.5	[0.15, 0.25]
c4.2xlarge	8	15	[0.35, 0.45]
r4.xlarge	4	30.5	[0.22, 0.32]
r4.2xlarge	8	61	[0.48, 0.58]
m4.xlarge	4	16	[0.15, 0.25]
m4.2xlarge	8	32	[0.35, 0.45]
m4.4xlarge	16	64	[0.75, 0.85]

We have conducted extensive simulations to evaluate the performance of our proposed multi-deadline time-sensitive VM provisioning schemes, and compared them against with the schemes that consider only a single deadline. In our experiments, we consider 8 different types of VMs, and their configurations are derived from amazon EC2 instance and are shown in Table 3. The table also shows the average bid per time unit for each VM.

To obtain the real workload trace data, we installed Openstack (version Mitaka) on a cluster of 10 compute nodes and run some applications from the NAS Parallel Benchmark suit on these VMs and collected their execution times. After collecting the runtime information for the benchmarks, we use them to drive our simulator to emulate different system loads. Table 5 shows the runtimes of some NPB benchmarks for their OMP versions on all the VMs types where empty cells indicates unsuccessful runs. Their runtime for the MPI versions are shown in Table 6. We run the MPI versions on all the VM types with different VM counts and under single node and multiple node configurations. Only few such combinations are shown in Table 6.

TABLE 4: Different clusters for a cloud system

Cluster	(CPUs, Memory(GB))	#Nodes
HTC-Small	(2, 4)	2
	(4, 8)	2
HTC-Medium	(2, 4)	4
	(4, 8)	2
	(8, 32)	2
HTC-Large	(2, 4)	8
	(4, 8)	4
	(8, 32)	2
	(16, 64)	2

Request Generation: User requests are generated from the runtime trace data in Table 5 and Table 6. For each request, we randomly pick one application from MPI or OMP benchmarks and use the associated VM type, VM counts, execution times and average bids. The arrival time is randomly generated between two consecutive interval times. The request is considered for allocation in the next interval.

For the execution time of the request, 1 mins (60 seconds) of VM setup/tear-down overhead is added to the execution time of the application. The first level deadline of each request is set as 2 to 4 times of its required execution time, and other levels are generated from an exponential growth function. For the bids of requests, we generate the first level bid by randomly selecting a value from the range showed in Table 3 and multiplying it with the VM count and the execution times. For the next level bids, an exponential decay function is used. For simplicity, we assume that each user request has 3 levels of $\langle$deadline, bid$\rangle$.

We consider cloud systems with different amounts of resources, where the heterogeneous clusters are shown in Table 4. For a cloud system with given cluster configuration, we generate requests to emulate the average system loads from $80\% - 140\%$. We run the simulation for 1000 minutes and use a fixed interval size $T = 5$ mins (if not specified otherwise). For each interval, the system load is set between $+/-40\%$ of the average system load.

Expedite and Delayed Services: Multiple deadline information of a request can be exploited in two different ways. If we think the last level $\langle$deadline, bid$\rangle$ as the base, then the earlier deadlines will indicate that the user is willing to pay higher bid values if they are served early. When we do such prioritization and allocation, we call it the *Expedite Service*. This is similar to the idea of a postal service where users can pick express delivery over the regular one by paying higher. On the other hand *Delayed Service* considers the first level $\langle$deadline, bid$\rangle$ as the base and extended deadlines represent how far the requests can be delayed. In this case, a user provides lower bids When the request is delayed. For both of the service types, the bids are the measure of how well the system can serve the user requests in terms of the response time.

Based on the service type, the scheme decides whether the last or the first level of $\langle$deadline, bid$\rangle$ to be considered as the base.

4.1. Performance for Expedited Service

To emulate the *Expedited Service*, we assume that the last deadline is the base, which will be exploited by the single-deadline based schemes (such as EDF, TSRF and TSRF/DR [32]). The other two earlier deadlines along with the associated larger bids will be exploited by our proposed slope-based M-TSRF and M-TSRF/DR schemes, which are denoted as S-M-TSRF and S-M-TSRF/DR, respectively.

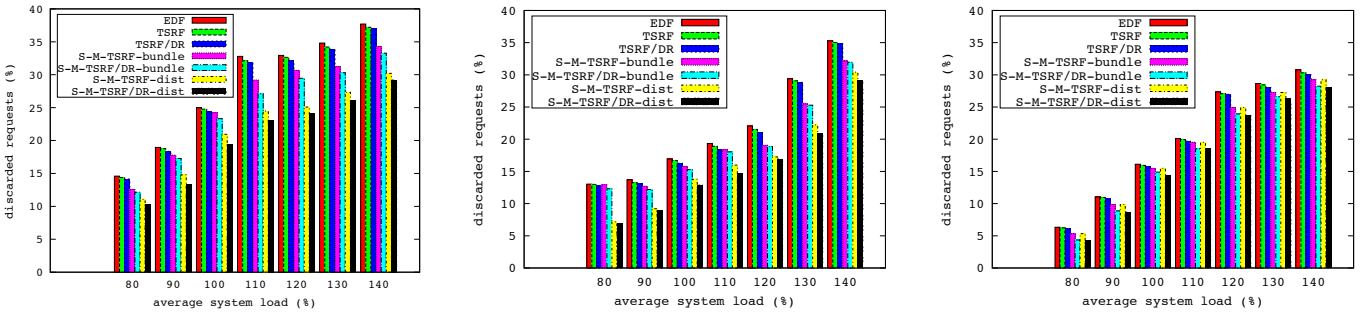
Figure 3 shows the discarded requests for the considered schemes under different system loads, where the Figures 3a, 3b and 3c are for the cloud systems with large, medium and small clusters, respectively. Here, we report the ratio of the number of discarded requests over the total number of generated requests. Clearly, as system loads become higher, more requests are discarded under all schemes. With only the last deadline being considered, we can see that

TABLE 5: Runtimes (min) of some NPB OMP benchmarks

Name	m4.xlarge	m4.2xlarge	m4.4xlarge	r4.xlarge	r4.2xlarge	c4.large	c4.xlarge	c4.2xlarge
BT	98.37	60.05	48.55	97.12	60.55	-	-	61.56
CG	-	30.94	21.72	58.01	30.39	-	-	-
EP	15.38	8.31	4.97	15.44	8.27	29.55	15.40	8.28
IS	-	-	0.83	-	1.35	-	-	-
LU	88.79	89.24	87.09	98.65	85.19	-	-	85.89
MG	-	6.54	5.74	8.08	6.12	-	-	-
SP	111.37	106.06	83.98	99.34	102.62	-	-	96.50
UA	67.38	56.79	43.32	78.83	51.43	-	-	51.37

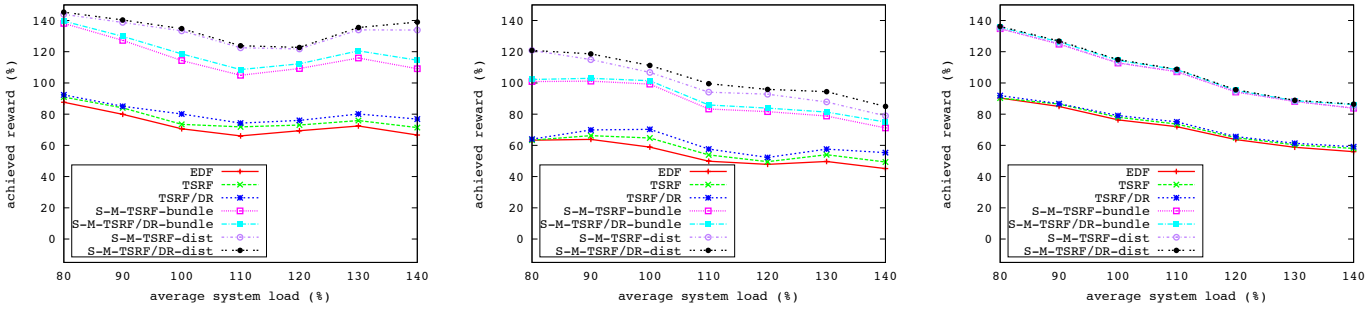
TABLE 6: Runtimes (min) of some NPB MPI benchmarks

Name	m4.xlarge(4,0)	m4.xlarge(2,2)	r4.2xlarge(2,0)	r4.2xlarge(1,1)	c4.large(8,0)	c4.large(4,4)	c4.xlarge(4,0)	c4.xlarge(2,2)
BT	73.75	57.54	85.14	79.83	69.61	82.41	74.29	75.68
CG	29.76	34.62	29.22	27.04	25.45	34.20	28.47	33.23
EP	8.07	6.39	8.05	7.29	7.51	8.43	7.48	7.92
LU	49.67	44.27	47.75	45.44	44.14	50.77	43.11	53.35
MG	5.29	4.60	5.55	5.28	5.06	6.43	4.79	5.12
SP	63.86	51.80	75.13	66.88	56.48	67.74	60.39	65.71



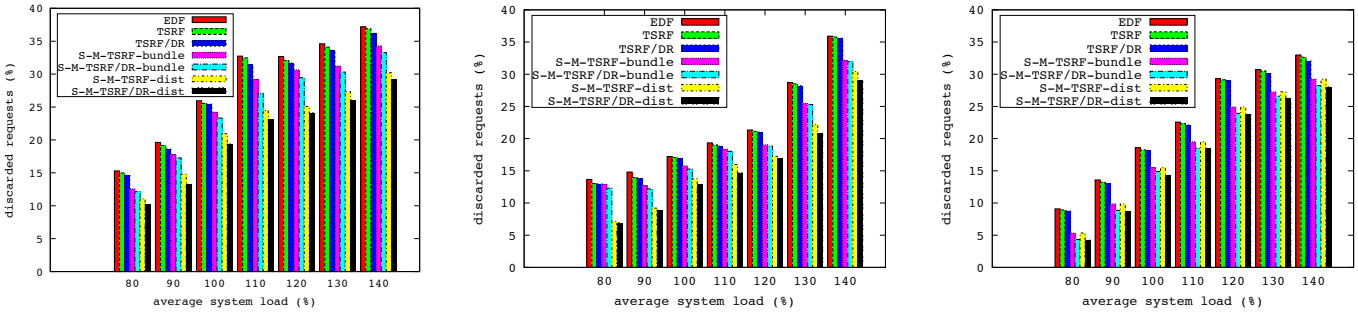
a. large cluster b. medium cluster c. small cluster

Figure 3: Discarded requests (%) for expedited service with $T = 5$ mins



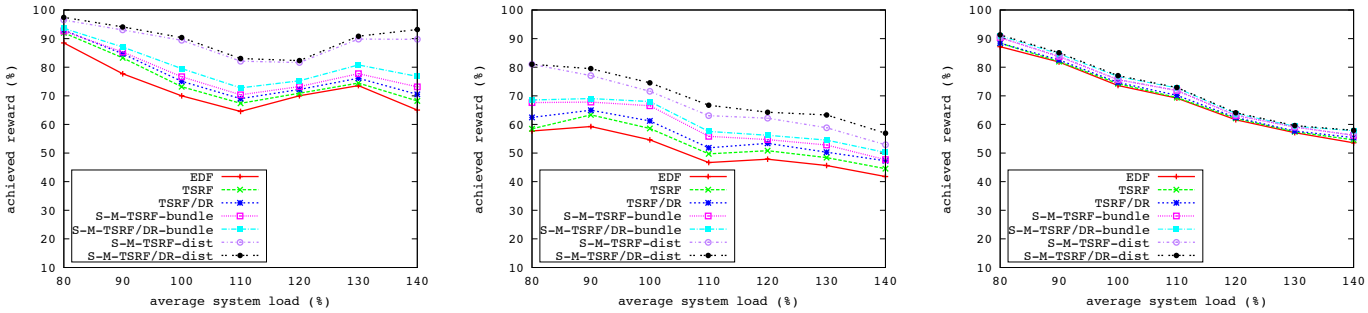
a. large cluster b. medium cluster c. small cluster

Figure 4: Achieved system benefit (%) for expedited service with $T = 5$ mins



a. large cluster b. medium cluster c. small cluster

Figure 5: Discarded requests (%) for degraded services with $T = 5$ mins



a. large cluster b. medium cluster c. small cluster

Figure 6: Achieved system benefit (%) for degraded services with $T = 5$ mins

both the single-deadline based TSRF and TSRF/DR schemes performs better than EDF with less requests being discarded, which is consistent with the existing results [32].

By considering the multiple (reduced) deadlines, our proposed slope-based multiple deadline schemes can reduce the discarded user requests, especially for higher system loads. Similar as the single-deadline based schemes, when the dominant resources are considered, less number of user requests are discarded. Moreover, compared to the bundled mapping where all VMs of a request need to be allocated to a single computing node, the distributed mapping has more flexibility when choosing computing nodes and thus has higher probability of successfully allocating the requests. As a result, the distributed mapping generally perform better than its counterpart with bundled mapping. Compared to the single deadline EDF scheme, our proposed multiple deadline schemes can reduce the discarded requests up to 8%.

Figure 4 further show the achieved system benefits as the ratio of the accumulated bids for the successfully served requests over the total last bids for all requests. First, for the single-deadline based schemes, the results are similar to those reported in our previous study [32]. Note that, for the proposed multiple deadline based schemes, a request can be served before its expedited deadline and higher bid than the last one can be achieved. Therefore, we can see that, the proposed schemes that consider multiple deadlines can achieve higher than 100% system benefit, especially for cloud systems with larger clusters of more resources. For the case of large cluster, the achieve system benefit of the proposed multiple deadline schemes is almost double of those achieved by the single deadline based schemes.

4.2. Performance for Degraded Service

Next, we consider the case of *degraded services* for requests. That is, for each request, we consider the first pair of deadline and bid as the baseline, which is utilized by the single deadline based schemes. The other two larger deadlines and smaller bids correspond to degraded services that can be accepted by the user, and will be exploited by our proposed schemes that consider multiple deadlines. Note that, a user request will be discarded by the single deadline based schemes if it can not be served by its first deadline. However, under the multiple deadline scheme, it will be discarded only if it can not be served before its latest deadline.

Figure 5 shows the discarded requests under all schemes for different system loads in cloud systems with large, medium and small clusters, respectively. Again, as system load increases, more requests are discarded by all schemes. For the single deadline based schemes, the TSRF and TSRF/DR perform better than EDF, which is consistent with our previous study [32].

When the extended deadlines of requests are considered, the proposed multiple deadline based schemes can further decreases the number of discarded requests, which is similar

to the case of expedited services. The performance difference between the proposed schemes and the single deadline based schemes becomes larger for cloud systems with more resources. Again, the schemes that consider distributed mapping have more more flexibility and thus have higher probability of successfully allocating the requests. Therefore, they perform better than those with bundled mapping where all VMs of a request need to be allocated together.

Figure 6 further shows the percentage of the achieved system benefits for all schemes. Since the baseline is the first pair with the largest bid for all requests, the achieved benefits are all less than 100%. However, by reducing the number of discarded requests, the proposed multiple deadline based schemes achieve more system benefit than those of single deadline based schemes, and the difference can be as high as 30%. But the difference diminishes for cloud systems with small cluster of less resources.

4.3. Impact of Interval Length on Performance

Interval length plays an important role in allocating VMs to the deadline-critical requests. Smaller interval length improves the performance by serving more user requests and achieving more profit. However it invokes the allocation schemes more frequently resulting in increased computational overhead. On the other hand, larger intervals reduces overhead but can miss deadlines of a large number of requests. That's why finding an optimal interval length is important.

We run simulation on the same request sets with different interval lengths to see how they impact the performance of our schemes. Figure 7 and Figure 8 show the percentage of discarded requests and achieved profit for $T = 5, 10$ and 20 mins under 120% system loads for *Expedite Service* respectively. With the increase of the interval length, missed request counts can reach upto 60%, 70% and 80% for large, medium and small clusters. As a result, achieved profit (%) gets as low as 20% for $T = 20$ mins. Similar trend can be seen for the *Delayed Service* in Figure 9 and Figure 10.

5. Conclusion

To support flexible timing requirements of user requests (for both expedited and degraded services), we consider requests with multiple deadlines and investigate time sensitive VM provisioning schemes when such requests run on resource constrained cloud systems. Specifically, we consider user requests that have multiple deadline-bid pairs to represent their flexible service requirements. Based on the rate of bid decreases for larger deadlines, we propose a *Slope-based* technique and integrate it with Time-Sensitive Resource Factor (TSRF) with Dominant Resource (DR) to prioritize those requests. Moreover, we extend Euclidean-distance based mapping by considering to allocate the multiple VMs of a user request to only one or multiple computing nodes, which are denoted as *Bundled* vs. *Distributed* mappings, respectively. Using execution trace data from real applications, we evaluate the proposed schemes with extensive simulations. The evaluation results show that, compared to the single deadline based schemes, the proposed multiple deadline based VM provisioning schemes and distributed mapping can significantly reduce the number of discarded requests, improve resource utilization and system benefit (up to 30%), especially for cloud systems with more resources and higher system loads.

References

[1] V. V. Vinothina, R. Sridaran, and P. Ganapathi, "A survey on resource allocation strategies in cloud computing," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 3, no. 6, 2012.

[2] M. Bjorkqvist, L. Y. Chen, and W. Binder, "Opportunistic service provisioning in the cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 237–244.

[3] F. Wuhib and R. Stadler, "Distributed monitoring and resource management for large cloud environments," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE, 2011, pp. 970–975.

[4] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner *et al.*, "Integrated capacity and workload management for the next generation data center," in *ICAC08: Proceedings of the 5th International Conference on Autonomic Computing*, 2008.

[5] S. Kim and Y. Kim, "Application-specific cloud provisioning model using job profiles analysis," in *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE, 2012, pp. 360–366.

[6] W.-T. Su and S.-M. Wu, "Node capability aware resource provisioning in a heterogeneous cloud," in *2012 1st IEEE International Conference on Communications in China (ICCC)*. IEEE, 2012, pp. 46–50.

[7] G. Le, K. Xu, and J. Song, "Dynamic resource provisioning and scheduling with deadline constraint in elastic cloud," in *2013 International Conference on Service Sciences (ICSS)*. IEEE, 2013, pp. 113–117.

[8] Y. Guo, P. Lama, J. Rao, and X. Zhou, "V-cache: Towards flexible resource provisioning for multi-tier applications in iaas clouds," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 88–99.

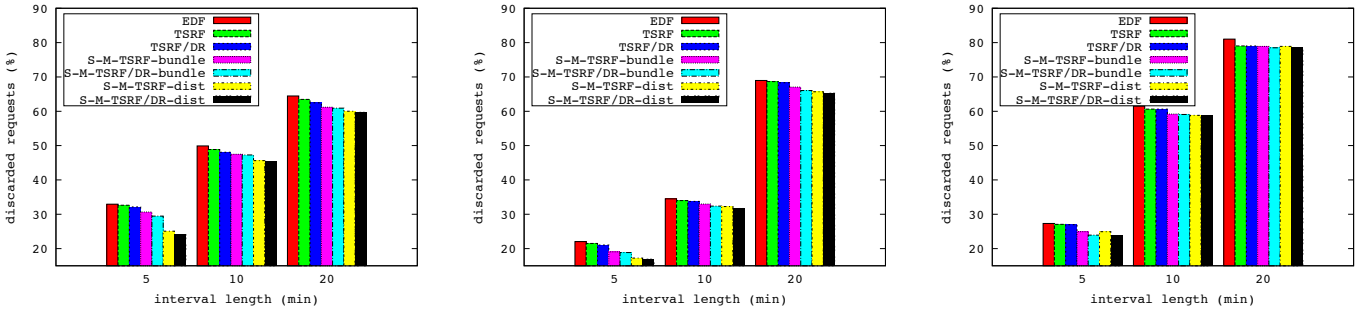
[9] C. S. Pawar and R. B. Wagh, "Priority based dynamic resource allocation in cloud computing," in *Cloud and Services Computing (ISCOS), 2012 International Symposium on*. IEEE, 2012, pp. 1–6.

[10] S. Zaman and D. Grosu, "Combinatorial auction-based mechanisms for vm provisioning and allocation in clouds," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 729–734.

[11] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic application container: A lightweight approach for cloud resource provisioning," in *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*. IEEE, 2012, pp. 15–22.

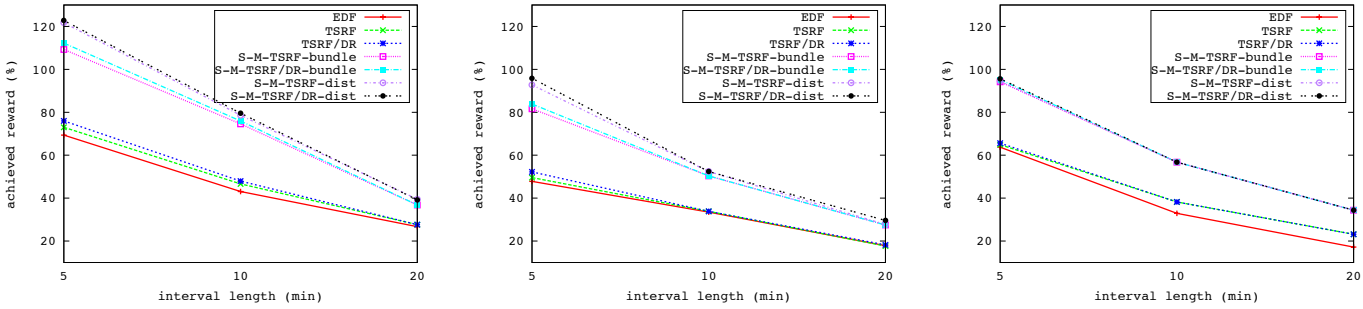
[12] B. B. Nandi, A. Banerjee, S. C. Ghosh, and N. Banerjee, "Dynamic sla based elastic cloud service management: A saas perspective," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 60–67.

[13] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 304–307.



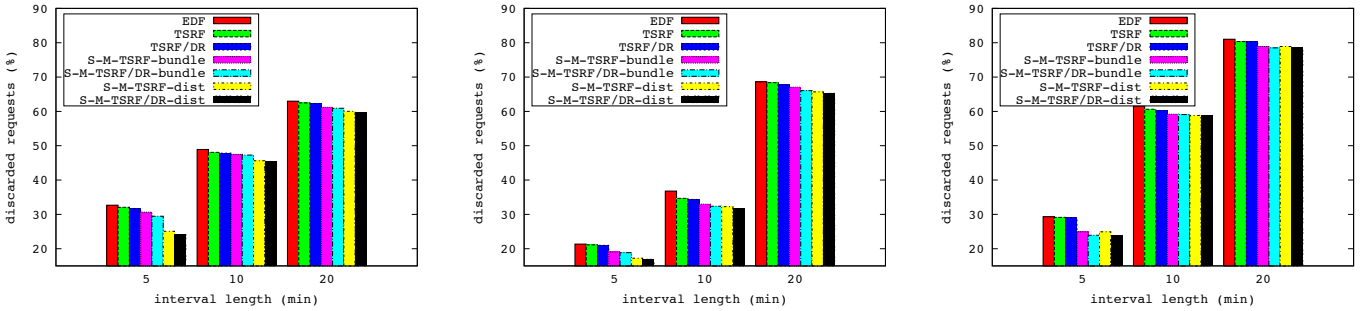
a. large cluster b. medium cluster c. small cluster

Figure 7: Discarded requests (%) for expedite service with 120% load



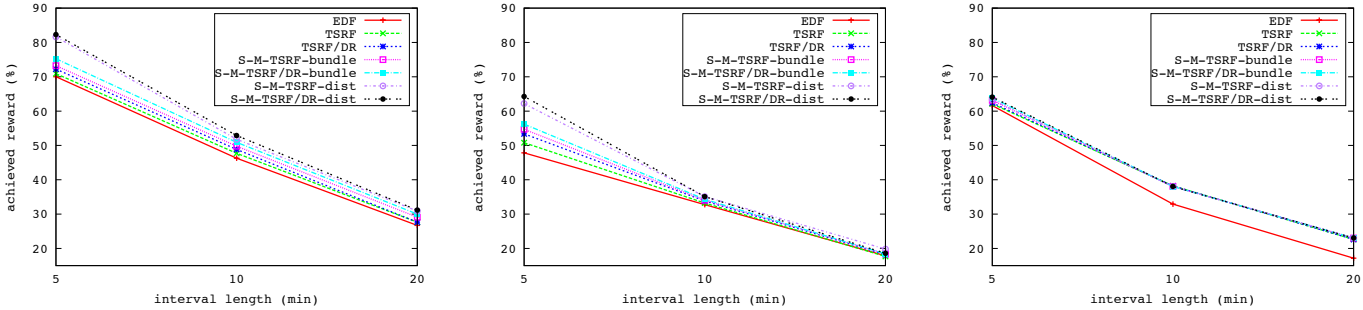
a. large cluster b. medium cluster c. small cluster

Figure 8: Achieved system benefit (%) for expedite service with 120% load



a. large cluster b. medium cluster c. small cluster

Figure 9: Discarded requests (%) for delayed service with 120% load



a. large cluster b. medium cluster c. small cluster

Figure 10: Achieved system benefit (%) for delayed service with 120% load

[14] G. Feng, S. Garg, R. Buyya, and W. Li, "Revenue maximization using adaptive resource provisioning in cloud computing environments," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE Computer Society, 2012, pp. 192–200.

[15] C. Tian, Y. Wang, F. Qi, and B. Yin, "Decision model for provisioning virtual resources in amazon ec2," in *Proceedings of the 8th International Conference on Network and Service Management*. International Federation for Information Processing, 2012, pp. 159–163.

[16] S. Rizou and A. Polyviou, "Towards value-based resource provisioning in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 2012, pp. 155–160.

[17] L. Shi, B. Butler, D. Botvich, and B. Jennings, "Provisioning of requests for virtual machine sets with placement constraints in iaas clouds," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 499–505.

[18] B. Javadi, J. Abawajy, and R. Buyya, "Failure-aware resource provisioning for hybrid cloud infrastructure," *Journal of parallel and distributed computing*, vol. 72, no. 10, pp. 1318–1331, 2012.

[19] B. Javadi, P. Thulasiraman, and R. Buyya, "Enhancing performance of failure-prone clusters by adaptive provisioning of cloud resources," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 467–489, 2013.

[20] B. Javadi, J. Abawajy, and R. O. Sinnott, "Hybrid cloud resource provisioning policy in the presence of resource failures," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 2012, pp. 10–17.

[21] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, 2011.

[22] G. Liu, H. Shen, and H. Wang, "Deadline guaranteed service for multi-tenant cloud storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2851–2865, 2016.

[23] D. Li, C. Chen, J. Guan, Y. Zhang, J. Zhu, and R. Yu, "Dcloud: deadline-aware resource allocation for cloud computing jobs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2248–2260, 2016.

[24] Y. Xiang, B. Balasubramanian, M. Wang, T. Lan, S. Sen, and M. Chiang, "Self-adaptive, deadline-aware resource control in cloud computing," in *Self-Adaptation and Self-Organizing Systems Workshops (SASOW), 2013 IEEE 7th International Conference on*. IEEE, 2013, pp. 41–46.

[25] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya, "Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 58–65, 2012.

[26] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh, "Spruce: A system for supporting urgent high-performance computing," *Grid-Based Problem Solving Environments*, pp. 295–311, 2007.

[27] "Amazon-emr, http://aws.amazon. com/cn/elasticmapreduce/?hp=tile," 2014.

[28] "Hadoop-rackspace, https://devel- oper.rackspace.com/databases/hadoop," 2014.

[29] M. Mao, J. Li, and M. Humphrey, "Cloud Auto-scaling with Deadline and Budget Constraints," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010.

[30] M. Mao and M. Humphrey, "Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[31] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds."

[32] R. Begam, W. Wang, and D. Zhu, "Timer-cloud: Time-sensitive vm provisioning in resource-constrained clouds," in *Cloud Computing; in submission; A preliminary version of this work appeared in HPCC 2015.*, May 2017.

[33] "Amazon ec2 instances, https://aws.amazon.com/ec2/instance-types/," March 2017.

[34] L. Mashayekhy, M. Nejad, and D. Grosu, "A ptas mechanism for provisioning and allocation of heterogeneous cloud resources," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[35] M. Nejad, L. Mashayekhy, and D. Grosu, "Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 594–603, Feb 2015.

[36] W. Wang, B. Liang, and B. Li, "Multi-resource fair allocation in heterogeneous cloud computing systems," *Parallel and Distributed Systems, IEEE Transactions on*, 2014.