Online Allocation of Virtual Machines in a Distributed Cloud

Fang Hao Murali Kodialam T. V. Lakshman Sarit Mukherjee
Bell Laboratories, Alcatel-Lucent
New Jersey, USA

***Abstract*—One of the primary functions of a cloud service provider is to allocate cloud resources to users upon request. Requests arrive in real-time and resource placement decisions must be made as and when a request arrives, without any prior knowledge of future arrivals. In addition, when a cloud service provider operates a geographically diversified cloud that consists of large number of small data centers, the resource allocation problem becomes even more complex. This is due to the fact that resource request can have additional constraints on data center location, service delay guarantee, etc. In this paper, we propose a generalized resource placement methodology that can work across different cloud architectures, resource request constraints, with real-time request arrivals and departures. The proposed algorithms are online in the sense that allocations are made without any knowledge of resource requests that arrive in the future, and the current resource allocations are made in such a manner as to permit the acceptance of as many future arrivals as possible. We derive worst case competitive ratio for the algorithms. We show through experiments and case studies the superior performance of the algorithms in practice.**

I. Introduction

A Cloud Service Provider (CSP) usually operates multiple data centers that are placed in different geographic regions. Cloud users request for different compute resources (i.e., CPU, memory, disk) and pay the CSP as they use the resources. One of the major problems that a CSP must handle when a request arrives at the cloud is to provision and place the resource request at the "right place" so that not only the needs for the request in question are satisfied, but also the placement decision keeps enough room for admission of future requests. Usually the requests come in the form of virtual machine (VM). A CSP makes different classes of VM configurations available to the cloud users. Each VM configuration consists of specific amount of CPU, memory and disk resource. It is the job of the placement algorithm to allocate the VMs in the best data center, in the most appropriate rack and host within the data center [1].

In this paper we propose resource allocation algorithms that are suited for use by CSPs that operate a large number of data centers where resources can be allocated. Cloud systems with a large number of data centers have the advantage of being closer to the cloud users and provide more location diversity. Intelligent placement of resources in a geographically diversified cloud system can lead to lower access latencies for the cloud users and lower bandwidth costs for the CSP. A geographically diversified cloud system has also been referred to as Distributed Cloud [1], [12] in the literature. Service providers with a large wide area networking footprint can build such systems to provide cloud services. Note, however, due to size limitation of individual data centers, resource allocation in a distributed cloud becomes more challenging than traditional cloud.

In a distributed cloud there are more choices where a requested resource as there are more data centers. Placement of resources cannot be done in isolation without considering the ways to interconnect them using the network resources. Moreover, cloud users can specify various types of connectivity constraints for accessing the cloud resources. The constraints can be the connectivity among the resources or between a cloud resource and the user's point-of-presence. A few examples are listed below:

- Geographic location preferences where a resource can be placed: Having resources close to a user location can yield lower response time whereas putting them far away may be suitable for disaster recovery.
- Cost limits on the resource usage: A user may choose to place resources in a cheaper data canter even if it is far from its location.
- Network distance bounds: A user can specify the distance between different resources or the distance between a cloud resource and the user's point-of-presence in terms of hop counts, delay, bandwidth etc.

The distinguishing features of our solution are as follows:

Revenue optimization vs. performance: The CSP is interested in maximizing its revenue from cloud services, which implies that the algorithm admits as many requests as it possibly can. On the other hand the user is interested in getting the best performance from the cloud, which can be easily achieved if the cloud is not overcrowded. Our algorithm is judicious in determining which data center to place new requests in.

Produce best solution with unknown future: Resource requests arrive one at a time to the CSP and leave when the job is completed. The algorithm works on placing the request based on the current work distribution within the cloud. Had the future requests be known to the placement algorithm, it would have been easier for the algorithm to come up with the "best" placement for the current request and also make enough room for requests arriving in the future. However, the

978-1-4799-3360-0/14/$31.00

algorithm cannot plan for the future placements or change the past placement decisions, but works competitively when the requests are not known a priori.

Be flexible in accommodating different resource constraints: We allow the user to specify resource constraints in various ways. Below we describe a few example cases:

- VM location: There are various ways a user can specify the location of a VM in the distributed cloud, e.g., the data center(s) suitable for the request, or in a data center that is certain distance away from a location, delay from a certain location, etc.
- VM duration: The request can specify at which time slots the VM is needed for the job. The times slots can be continuous, discontinuous or periodic.
- Inter-VM distance: The request can specify groups of required VMs and specify the relationship among the groups, e.g., close-by, far-off, in the same data center, etc. This type of constraints helps in properly configuring VMs for certain jobs, and can also be used for achieving reliability and availability.
- CSP's cost structure: The service provider can have a cost structure or policy for resource allocation that can be specified to the algorithm.

Generate results in real-time: Majority of the placement problems turn out to be NP-hard problems. The proposed methodology produces competitive solutions with varied user specific constraints.

In this paper we propose a generalized methodology for online resource placement in cloud systems. The methodology can handle resource allocation in both traditional and distributed cloud systems. These problems are NP-hard in general, and we propose approximation algorithms for joint allocation of different resources (e.g., CPU, memory, disk, etc.) to successfully instantiate a requested VM (Sections II and III). We derive worst case competitive ratios for the algorithms (Section IV). Distributed cloud brings in additional choices/constraints for resource placement, and we show how that can be handled in our methodology (Section V). Using experiments and case study we show that our algorithms perform far superior to the worst case in all practical situations (Section VI).

II. Problem Definition

We now formally define the problem of VM allocation in a distributed cloud and outline an online methodology to allocate VMs to different data center locations. We assume a time slotted system and t indexes the times slots. Assume that we have a distributed cloud with n data center locations. There are a set of R resources at each of the locations. We use $|R|$ to denote the total number of resources. Typical resources include CPU, memory and storage disk. We assume that the amount of resource $r \in R$ available at location i at time t is given by $c(i,r,t)$ and is known to the allocator. We assume that there are K classes of VMs. The amount of resource r required to host a VM of class k is given by $g(k,r)$. Request arrivals to the system are indexed by ℓ. Each request is for a set of VMs. Let $T(\ell)$ represent the set of time periods for which the VMs are needed, and $L(\ell)$ denote the total number of time slots in $T(\ell)$. Associated with each request is a set of feasible configurations $\mathcal{P}_\ell$. The set of feasible configurations may be defined implicitly. For a configuration $P \in \mathcal{P}_\ell$, let $A(P,\ell,i,k)$ represent the number of VMs of class k that are at location i. Let $N(\ell,k) = \sum_i A(P,\ell,i,k)$ represent the total number of VMs of class k that are needed to satisfy request ℓ. The amount of resource r required at location i if configuration P is used is $A(P,\ell,i,k)g(k,r)$. The decision variable for request ℓ is a binary variable X_P^ℓ which is set to one if configuration $P \in \mathcal{P}_\ell$ is used for request ℓ. If $X_P^\ell = 0$ for all $P \in \mathcal{P}_\ell$, then request ℓ is not instantiated by the cloud system. We assume that instantiating one VM of type k for one time period generates a revenue of $p(k)$ for the cloud service provider. The objective of the cloud service provider is to place the VMs so as to maximize its total revenue. Ideally, we want the VM location decisions to be made in an online manner without any knowledge of future requests.

A. Complete Information Scheduler

Before considering the online VM allocation problem, we first consider the offline version of the same problem. In the offline version of the problem, we assume that we know all the demands ahead of time. The objective of the offline VM allocator is to determine the valid mapping P for each time period that maximizes the overall number of accepted VMs. As defined before, $X_P^\ell = 1$ if mapping $P \in \mathcal{P}_\ell$ is used for request ℓ and zero otherwise. The problem of maximizing the number of VMs can be written as

$$\max \sum_\ell L(\ell) \sum_k N(\ell,k)p(k) \sum_{P\in\mathcal{P}_\ell} X_P^\ell$$

$$\sum_{P\in\mathcal{P}_\ell} X_P^\ell \leq 1 \quad \forall \ell \tag{1}$$

$$\sum_{\ell:T(\ell)\ni t} \sum_k A(\ell,P,i,k)g(k,r)X_P^\ell \leq c(i,r,t) \quad \forall i,r,t \tag{2}$$

$$X_P^\ell \in \{0,1\} \quad \forall P,t \tag{3}$$

For a given request ℓ, note that $\sum_k N(\ell,k)p(k)$ represents the total revenue generated across all classes of VMs for a given time slot. Since request ℓ is for $L(\ell)$ time slots, the total revenue is the product of these two quantities. This is independent of the configuration $P \in \mathcal{P}_\ell$ used. Equation [1] ensures that at most one mapping is used for each request. Equation [2] ensures that the total amount of resource r consumed in time period t at location i does not exceed the capacity $c(i,r,t)$. Note that a set of VMs can be instantiated at a location only if *all* resources needed for the VMs are available at the location. We consider the linear programming relaxation of the above problem where we set $0 \leq X_P^l \leq 1$. The upper bound $X_P^l \leq 1$ is implied by Equation [1] and can therefore be eliminated from the formulation. We now write the dual to the above linear programming relaxation as

$$\min \sum_\ell \pi(\ell) + \sum_i \sum_r \sum_t c(i,r,t)\delta(i,r,t)$$

$$\begin{aligned}\pi(\ell) &\geq \sum_k N(\ell,k)p_k - \\ &\quad \sum_{i,r,k,t} A(P,\ell,i,k)g(k,r)\delta(i,r,t)\ \forall P \in \mathcal{P}_\ell\ \forall \ell \\ \pi(\ell) &\geq 0\ \forall \ell \\ \delta(i,r,t) &\geq 0\ \forall i,r,t\end{aligned}$$

Since the inequality for $\pi(\ell)$ holds for all mappings P, we can set

$$\begin{aligned}\pi(\ell) &= \max_{P\in\mathcal{P}_\ell} L(\ell)\sum_k N(\ell,k)p(k) - \\ &\quad \sum_{i,r,k,t} A(P,\ell,i,k)g(k,r)\delta(i,r,t) \\ &= L(\ell)\sum_k N(\ell,k)p(k) - \\ &\quad \min_{P\in\mathcal{P}_\ell} \sum_{i,k} A(P,\ell,i,k)\sum_r g(k,r)\sum_t \delta(i,r,t).\end{aligned}$$

By linear programming duality, the optimal solutions to the primal and the dual are equal. Moreover, any feasible solution to the dual will produce an upper bound to the primal optimal solution. Clearly, the information (i.e., all the request arrivals) for the primal or the dual will not be known ahead of time. Therefore the objective is to develop an online algorithm that does not assume any information about the future. The performance of the online algorithm will be compared to the optimal offline algorithm.

B. Performance Guarantee

Let Z^* denote the optimal solution to the linear programming problem. This is the maximum revenue that is achieved by the offline algorithm. Let Z_{ON} represent the solution obtained by the online algorithm. Note that $Z_{ON} \leq Z^*$ since the online algorithm can at best match the offline optimal solution. We say that the online algorithm is β-competitive if there exists a $\beta \leq 1$ such that

$$Z_{ON} \geq \beta\ \ Z^*$$

for *all* possible inputs. In general, we would like to get a competitive ratio that is close to one. This is possible in the case of some simple online optimization problems. For the more complex problems like the VM allocation problem, we can show that a constant factor competitive algorithm is not possible. Instead we define the notion of $[c_1, c_2]$ competitive algorithm [3].

Definition 2.1: An online algorithm is $[c_1, c_2]$-competitive if the online algorithm achieves a profit of at least $c_1 \leq 1$ of the optimum offline solution while ensuring that the utilization of any resource does not exceed $c_2 \geq 1$.

For example an online algorithm is $[0.5, \log n]$ competitive if the online algorithm gets at least 0.5 of the revenue of the offline algorithm while none of the resources are utilized more than $\log n$. We ideally want none of the resources to be utilized more than one. For many difficult online optimization problems, this logarithmic capacity violation is the best possible guaranteed bound. One way of getting around this excess utilization is to scale down the capacities by $\log n$ so that the capacity will not be exceeded. This may be too pessimistic in practice. Instead, we show how to modify the algorithm to get good online allocation performance in practice.

III. Online VM Allocation Algorithm

At each resource request, the online algorithm constructs a primal solution that is an allocation and a corresponding dual solution that represents an upper bound on the optimal solution value. This follows the general procedure outlined in [4]. However, unlike the problem in [4], in our problem, requests only stay for a finite amount of time. Moreover, each request generates demand for multiple resources. The primal and dual solutions are carefully constructed so that they are within a constant factor of each other. Since a feasible dual solution is an upper bound on the primal, we can show that the primal is within a constant factor of the optimal solution. The steps in the online allocation algorithm, *Online VM Allocator* are shown in Figure 1. The dual variables $\delta(i,r,t)$ are initialized to zero for all i,r,t. At each arrival, the VM allocator has to determine the mapping P that minimizes $\sum_{i,k} A(P,\ell,i,k)\sum_r g(k,r)\sum_t \delta(i,r,t)$. We can view $\sum_r g(k,r)\sum_t \delta(i,r,t)$ as the cost of having one unit of VM type k assigned to location i. Recall that one VM type k requires $g(k,r)$ units of resource r. We set $w(i,k) = \sum_r g(k,r)\sum_t \delta(i,r,t)$ as the cost of assigning one VM type k to location i. Note that $w(i,k)$ can be computed since the current values of $\delta(i,r,t)$ are known. The problem of determining the minimum mapping can be written as

$$\min_{P\in\mathcal{P}_\ell} \sum_{i,k} w(i,k)A(P,\ell,i,k).$$

The complexity of solving this problem depends on the set $\mathcal{P}_\ell$. In a later section we show how the algorithm can be modified if we can only get an approximate optimal solution. In the next section, we derive the approximation guarantee provided by the Online VM Allocator algorithm.

IV. Approximation Guarantee

The online algorithm constructs a primal and dual solution after each arrival. If the dual solution is feasible, then it provides an upper bound on the optimal solution. We show that the dual solution is feasible and that the primal and dual solutions constructed are within a constant factor of each other. Finally, we bound the constraint violation of the primal solution to show a $[c_1, c_2]$-competitive algorithm.

A. Dual Feasibility

Note that by picking the minimum weight mapping for arrival ℓ in Step 2, the dual solution is feasible. Since the value of $\delta(i,r,t)$ only increases, the solution will remain feasible subsequently.

Online VM Allocator

- Initialize $\delta(i,r,t) \leftarrow 0 \ \ \forall i,r,t$.
- For each request ℓ:
- 1. Compute the weight $w(i,k)$ of assigning one VM of type k to location i.

$$w(i,k) = \sum_r g(k,r) \sum_t \delta(i,r,t)$$

- 2. Find the minimum weight mapping P^* for the current arrival

$$v = \min_{P \in \mathcal{P}_\ell} \sum_{i,k} w(i,k) A(P,\ell,i,k).$$

$$P^* = \arg\min_{P \in \mathcal{P}_\ell} \sum_{i,k} w(i,k) A(P,\ell,i,k).$$

- 3. If $L(\ell) \sum_k N(\ell,k) p(k) - v \leq 0$ reject the request. Else
- 4a. Place request ℓ at P^*.
- 4b. Set

$$\begin{aligned} \delta(i,r,t) &\leftarrow \delta(i,r,t) \left[1 + \frac{\sum_k A(P^*,\ell,i,k) g(k,r)}{c(i,r,t)}\right] \\ &+ \frac{1}{e-1} \frac{\sum_k A(P^*,\ell,i,k) p(k)}{|R| \ c(i,r,t)} \end{aligned}$$

- 4c. Set

$$\begin{aligned} \pi(\ell) &= L(\ell) \sum_k N(\ell,k) p(k) - \\ &\sum_{i,r,k,t} A(P^*,\ell,i,k) g(k,r) \delta(i,r,t) \end{aligned}$$

Fig. 1. Description of Online VM Allocator

B. Primal to Dual Ratio

We first show that the values of $\delta(i,r,t)$ and $\pi(\ell)$ generated by the algorithm are dual feasible. Let P^* denote the maximum weight mapping for arrival ℓ. Let $\Delta\delta(i,r,t)$ denote the increase in $\delta(i,r,t)$. Let $\phi = \sum_{irt} c(i,r,t) \Delta\delta(i,r,t)$. Then

$$\begin{aligned} \phi &= \sum_{i,r,t,k} A(P^*,\ell,i,k) g(k,r) \delta(i,r,t) + \\ &\frac{1}{e-1} \sum_{i,r,t,k} \frac{A(P^*,\ell,i,k) p(k)}{|R|} \end{aligned}$$

Let $\theta = \pi(\ell) + \phi$. Then

$$\begin{aligned} \theta &= L(\ell) \sum_i \sum_k A(P^*.\ell,i,k) p(k) + \\ &\frac{1}{e-1} \sum_{i,r,t,k} \frac{A(P^*,\ell,i,k) p(k)}{|R|} \\ &= \sum_t \left[1 + \frac{1}{e-1}\right] L(\ell) \sum_k N(\ell,k) p(k) \end{aligned}$$

where we used the following fact:

$$\begin{aligned} \sum_{i,r,t,k} \frac{A(P^*,\ell,i,k) p(k)}{|R|} &= \sum_{r,t,k} \sum_i \frac{A(P^*,\ell,i,k) p(k)}{|R|} \\ &= \sum_{t,k} \sum_r \frac{N(\ell,k) p(k)}{|R|} \\ &= \sum_{t,k} N(\ell,k) p(k) \\ &= L(\ell) \sum_k N(\ell,k) p(k) \end{aligned}$$

Note that $\sum_i A(P^*,\ell,i,k) = N(\ell,k)$.

C. Primal Feasibility

Let $U_b(i,r,t)$ denote the utilization at location i for resource r at time slot t after arrival $\ell - 1$ is processed.

$$U_b(i,r,t) = \frac{\sum_{j=1}^{\ell-1} \sum_k A(P_j^*,j,k,t) g(k,r)}{c(i,r,t)}.$$

The numerator represents the total consumption of resource r at location i for time slot t that is allocated by all arrivals up to $\ell - 1$. We use P_j^* to denote the optimal mapping for arrival j. Similarly, we define $U_e(i,r,t)$ to be the utilization of resource r at location i at time t after arrival ℓ is processed. The only difference between $U_b(i,r,t)$ and $U_e(i,r,t)$ is that the summation in the numerator goes up to arrival ℓ. We want to show that

$$\delta(i,r,t) \geq \frac{e^{U_e(i,r,t)-1}}{e-1} \ \ \forall i,r,t$$

after arrival ℓ. We prove this by induction on the arrivals. Note that initially $\delta(i,r,t) = 0$ for all i,r,t and therefore the statement is true at the system start time. Assume that the inequality is satisfied up to arrival $\ell - 1$. We want to show that the statement is true after arrival ℓ. Let P^* denote the optimal mapping for arrival ℓ. Let $\delta_b(i,r,t)$ denote the value of $\delta(i,r,t)$ after arrival $\ell-1$ is processed and $\delta_e(i,r,t)$ denote the value of $\delta(i,r,t)$ after arrival ℓ is processed. Then

$$\begin{aligned} \delta_e(i,r,t) &= \delta_b(i,r,t) \left[1 + \frac{\sum_k A(P^*,\ell,i,k) g(k,r)}{c(i,r,t)}\right] \\ &+ \frac{1}{e-1} \frac{\sum_k A(P^*,\ell,i,k) p(k)}{|R| \ c(i,r,t)} \\ &\geq \frac{e^{U_b(i,r,t)} - 1}{e-1} \left[1 + \frac{\sum_k A(P^*,\ell,i,k) g(k,r)}{c(i,r,t)}\right] \\ &+ \frac{1}{e-1} \frac{\sum_k A(P^*,\ell,i,k) p(k)}{|R| \ c(i,r,t)} \\ &= \frac{e^{U_b(i,r,t)}}{e-1} \left(1 + \frac{\sum_k A(P^*,\ell,i,k) g(k,r)}{c(i,r,t)}\right) \\ &- \frac{1}{(e-1)c(i,r,t)} - \frac{\sum_k A(P,\ell,i,k) g(k,r)}{(e-1)c(i,r,t)} \\ &+ \frac{\sum_k A(P,\ell,i,k) p(k)}{(e-1)c(i,r,t)|R|} \end{aligned}$$

Assume that the profit $p(k)'s$ are scaled so that $p(k) \geq |R|g(k,r) \quad \forall r,k$. Let $\underline{P} = \min_k p(k)$ denote the minimum profit for any VM class , $\overline{G} = \min_{k,r} g(k,r)$ denote the maximum resource of any type that is consumed by any class of VM. Let $\underline{P} \geq |R|\overline{G}$, then we can write

$$\begin{aligned}\delta_e(i,r,t) &\geq \frac{e^{U_b(i,r,t)}}{e-1}\left(1+\frac{\sum_k A(P^*,\ell,i,k)g(k,r)}{c(i,r,t)}\right) \\ &\quad -\frac{1}{(e-1)c(i,r,t)} \\ &\geq \frac{e^{U_e(i,r,t)}-1}{e-1},\end{aligned}$$

where in the last step we assume that the amount of resource allocated by any given arrival to a particular location is a very small fraction of the total amount of resource available at that location, i.e.,

$$\frac{\sum_k A(P^*,\ell,i,k)g(k,r)}{c(i,r,t)} \to 0$$

and we can write

$$e^{\frac{\sum_k A(P^*,\ell,i,k)g(k,r)}{c(i,r,t)}} \approx \left(1+\frac{\sum_k A(P^*,\ell,i,k)g(k,r)}{c(i,r,t)}\right).$$

Let $\overline{P} = \max_k p(k)$, $\underline{G} = \min_{k,r} g(k,r)$ and $\overline{L} = \max_\ell L(\ell)$ denote the number of time periods that the longest job lasts. We know that $\pi(j) \geq 0$. Using this fact, we can write

$$\begin{aligned}0 &\leq L(\ell)\sum_k N(\ell,k)p(k) - \sum_{i,r,k,t} A(P,\ell,i,k)g(k,r)\delta(i,r,t) \\ &\leq \overline{LP}\sum_k N(\ell,k) - \underline{G}\sum_{i,r,k} A(P,\ell,i,k)\sum_t \delta(i,r,t) \\ &\leq \overline{LP}\sum_k N(\ell,k) - \underline{G}\delta(i,r,t)\sum_{i,r,k} A(P,\ell,i,k) \\ &\leq \overline{LP}\sum_k N(\ell,k) - \underline{G}|R|\delta(i,r,t)\sum_{i,k} A(P,\ell,i,k) \\ &= \overline{LP}\sum_k N(\ell,k) - \underline{G}|R|\delta(i,r,t)\sum_k N(\ell,k)\end{aligned}$$

Therefore

$$\delta(i,r,t) \leq \frac{\overline{LP}}{\underline{G}|R|}.$$

Since

$$\delta(i,r,t) \geq \frac{e^{U(i,r,t)}-1}{e-1}$$

we can write

$$U(i,r,t) \leq \log\left((e-1)\frac{\overline{LP}}{\underline{G}|R|}+1\right).$$

Therefore the maximum utilization of any resource is upper bounded by this quantity. Note that we have assumed that $\underline{P} \geq |R|\overline{R}$ when deriving the dual to primal ratio. Therefore

$$\frac{\overline{LP}}{\underline{G}|R|} \geq 1.$$

We state the result of the above three steps in the Theorem below.

Theorem 4.1: If the individual VM requests are small compared to the individual data center (i.e., locations) capacities, then the Online VM Allocator algorithm is $\left[\frac{e-1}{e}, \log\left((e-1)\frac{\overline{LP}}{\underline{G}|R|}+1\right)\right]$-competitive where $\overline{L} = \max_\ell L(\ell)$, $\overline{P} = \max_k p(k)$, $\underline{G} = \min_{k,r} g(k,r)$, and $|R|$ is the number of resources.

This theorem generalizes the result in [3]. There are several special cases that are of interest:

- In the case where all requests stay permanently, we do not need to time index the dual variables, and in that case the algorithm just maintains a dual variable $(\delta(i,r)$ for each location-resource combination. The algorithm will be $\left[\frac{e-1}{e}, \log\left((e-1)\frac{\overline{P}}{\underline{G}|R|}+1\right)\right]$-competitive.
- If in addition to the condition above, we have only one resource and if the profit associated with the resource is proportional to the amount of resource consumed (an example is when the profit associated with each VM is equal, and therefore the total profit is proportional to the number of VMs) then the algorithm is $\left[\frac{e-1}{e}, 1\right]$-competitive and there will be no capacity violation.
- Note that the dual variable $\delta(i,r,t)$ increases exponentially with that resource's utilization $U(i,r,t)$. Therefore $\delta(i,r,t) = O\left(e^{U(i,r,t)}\right)$. If instead we set $\delta(i,r,t) = O\left(C^{U(i,r,t)}\right)$ for some constant C, then as C becomes large, the algorithm just becomes a min-max utilization algorithm; allocation is made where the maximum utilization of any resource is minimized at the end of the allocation. This has been observed to be a good heuristic in practice but it does not have a competitive guarantee.

We next outline an implementation of the Online VM Allocator algorithm that takes several practical aspects into consideration.

D. Maintaining Feasibility

The algorithm as stated above can violate the capacity constraints. While this is the best possible bound achievable in theory, it will not be an implementable solution in practice. Therefore, we make the following modifications to the algorithm so that no capacity bounds are violated and we still have an upper bound. The guarantee on competitive ratio is sacrificed but we show that in practice, the performance of this technique is still very good. We define a mapping $P \in \mathcal{P}_\ell$ to be **feasible**, if assigning the current request according to mapping P does not result in capacity violation at any location i for any resource r. In other words, there should be a spare capacity of $\sum_k A(P,\ell,i,k)g(k,r)$ at location i for resource r. In order to keep the solution feasible, we modify Step 2-4 of the Online VM Allocator. This algorithm, referred to as the Feasible VM Allocator is shown in Figure 2

Note that the $\delta(i,r,t)$ is updated based on the minimum weight feasible $\tilde{P}$ but $\pi(\ell)$ is updated based on the minimum weight mapping P^*. Therefore the guarantee on the bounds will not be valid. However, note that we have a dual feasible solution and by construction a primal feasible solution. Therefore, we can still use the dual solution to get an upper bound on the optimal solution value. This will be referred to as

Feasible VM Allocator

- Initialize $\delta(i,r,t) \leftarrow 0 \ \ \forall i,r,t$.
- For each request ℓ:
- 1. Compute the weight $w(i,k)$ of assigning one VM of type k to location i.

$$w(i,k) = \sum_r g(k,r) \sum_t \delta(i,r,t)$$

- 2. Find the minimum weight **feasible** mapping $\tilde{P} \in \mathcal{P}_\ell$ for the current arrival.
- 3. Reject if no feasible allocation. Else
- 4a. Place request at $\tilde{P}$.
- 4b. Set

$$\begin{aligned}\delta(i,r,t) \quad \leftarrow \quad & \delta(i,r,t)\left[1 + \frac{\sum_k A(\tilde{P},\ell,i,k)g(k,r)}{c(i,r,t)}\right] \\ & + \frac{1}{e-1}\frac{\sum_k A(\tilde{P},\ell,i,k)p(k)}{|R|\ c(i,r,t)}\end{aligned}$$

- 4c. Set

$$\begin{aligned}\pi(\ell) \quad = \quad & L(\ell)\sum_k N(\ell,k)p(k) - \\ & \sum_{i,r,k,t} A(P^*,\ell,i,k)g(k,r)\delta(i,r,t)\end{aligned}$$

where $P^* \in \mathcal{P}_\ell$ is the minimum weight mapping.

Fig. 2. Description of Feasible VM Allocator

the Single Update Upper Bound (SUUB) in the experimental section.

E. Improving the Bound in Practice

Though the algorithm guarantees a worst case competitive ratio of $\frac{e-1}{e}$, in practice it is possible to improve the upper bound on the optimal solution. Note that we set

$$\begin{aligned}\pi(\ell) \quad = \quad & L(\ell)\sum_k N(\ell,k)p(k) - \\ & \sum_{i,r,k,t} A(P^*,\ell,i,k)g(k,r)\delta(i,r,t)\end{aligned}$$

and the bounds are computed based on this value of $\pi(\ell)$. As the value of $\delta(i,r,t)$ increases, it is possible to reduce the value of $\pi(\ell)$ further. Assume that P^*_ℓ is the optimal allocation of request ℓ. As more requests enter the system and these are assigned, the value of $\delta(i,r,t)$ for some (i,r,t) combinations will increase if there are allocations made to this (i,r,t) combination. Let $\delta_c(i,r,t)$ denote the current value of $\delta(i,r,t)$. We can now compute a new value of $\pi_c(\ell)$ as

$$\begin{aligned}\pi_c(\ell) \quad = \quad & L(\ell)\sum_k N(\ell,k)p(k) - \\ & \sum_{i,r,k,t} A(P^*_\ell,\ell,i,k)g(k,r)\delta_c(i,r,t).\end{aligned}$$

Note that $\pi_c(\ell) \leq \pi(\ell)$ and therefore provides a better upper bound on the offline optimal solution. Thus we can compute tighter upper bound in practice than the $\frac{e-1}{e}$ bound given in theory. The bound calculated using the above procedure is referred to as the Multiple Update Upper Bound in the experimental section.

Thus far, we have modeled the constraints specified by request ℓ abstractly as set of permitted mappings $\mathcal{P}_\ell$ of VM to locations. In the next section, we outline some more concrete examples of these constraints.

V. Constraints on the Allocation

When request ℓ enters the system, the constraints on the allocation of ℓ are specified as a set of permitted mappings $\mathcal{P}_\ell$. Since we have to solve the minimum weight allocation in order to determine the allocation of arrival ℓ, the complexity of allocation depends crucially on the structure of $\mathcal{P}_\ell$. Before we describe examples of constraints, we first outline some network information that can be used to specify feasible mappings. There is a network that interconnects the different cloud data center locations. We define the distance between two data center locations i and i' as $d(i,i')$. This distance can either be the physical distance, the length of the shortest path from i to i' in the underlying network or the estimated mean delay to transfer a packet from location i to location i'. This network information is used to specify constraints on VM allocation. We now describe several examples of constraints that might occur in practice:

- *Simple Location Restriction:* In the case of simple location restriction, each request comprises of a group of VMs perhaps of different types. These VMs have to be placed within a distance of $d_{\max}$ of an endpoint. This in turn can be translated into a subset of locations that are feasible. Request ℓ is constrained to be located in the set $S(\ell)$ of locations. An example of the set $S(\ell)$ is the set of all locations that are within a distance of Δ from some given location j, i.e.,

$$S(\ell) = \{i : \delta(i,j) \leq d_{\max}\}.$$

- *Location with Diversity Requirements:* The locations of the distributed data center are partitioned into different zones. These zones represent locations that do not have any common failure points. In this case of VMs with diversity requirements, request ℓ is for two sets of VMs. The constraint is that the two groups are in different zones.
- *Inter VM Distance Restriction:* In addition to the simple location restriction, there can be additional restrictions on the VMs associated with request ℓ. The most common restriction that we consider is specifying the maximum distance between pairs of VM locations. For example, there might be a constraint that specifies that the maximum distance between the location of VM set j and VM set j' of request ℓ is bounded by some maximum value.

In the first two cases, it is easy to solve the minimum weight mapping. If the minimum weight mapping problem is NP-hard

but there is a constant factor $\alpha \geq 1$ approximation algorithm to solve the problem, then the VM allocation algorithm can be modified as follows: (We only outline the relevant steps):

- 4b. Set

$$\delta(i,r,t) \leftarrow \delta(i,r,t)\left[1+\frac{\alpha\sum_k A(\tilde{P},\ell,i,k)g(k,r)}{c(i,r,t)}\right] + \frac{\alpha}{e^\alpha-1}\frac{\sum_k A(\tilde{P},\ell,i,k)p(k)}{|R|\ c(i,r,t)}$$

- 4c. Set

$$\pi(\ell) = \alpha L(\ell)\sum_k N(\ell,k)p(k) - \alpha\sum_{i,r,k,t} A(P^*,\ell,i,k)g(k,r)\delta(i,r,t)$$

where $P^* \in \mathcal{P}_\ell$ is the minimum weight mapping.

With this modification, we can show as in [8], that the competitive ratio is

$$\frac{e^\alpha}{\alpha e^\alpha - 1}$$

instead of $\frac{e}{e-1}$. In cases where there is no guaranteed approximation algorithm to solve the minimum weight mapping problem, some heuristic has to be used to solve the problem. If a lower bound can be computed for each step of the heuristic, this can be translated into an upper bound on the optimal solution to the offline problem. We now provide some performance results for the VM allocation algorithm.

VI. Experimental Results

We present some experimental results to show the performance of the algorithm in practice. The main objective is to check the performance of the VM allocation algorithm which maintains feasibility. Therefore the allocation algorithm always ensures that there is enough capacity before making an allocation. Recall that $\delta(i,r,t)$ is updated based on the optimal solution and not on the current assignment. For each problem, in addition to the primal solution we show three upper bounds on the feasible solution.

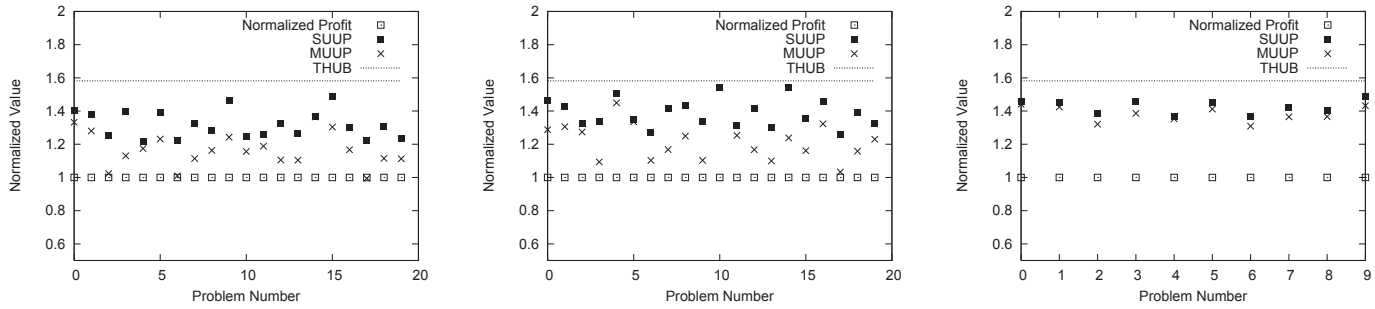
- The theoretically guaranteed value that is $\frac{e}{e-1}$ of the primal solution. Note that the theoretical guarantee holds only for the Online VM Allocator algorithm and not for the Feasible VM Allocator algorithm. We plot this value in the graph for reference. We call this the Theoretical Upper Bound (THUB) in the plot.
- The dual feasible solution where the $\pi(\ell)$ is only updated when request ℓ is allocated. The value of $\pi(\ell)$ is not recomputed. We call this Single Update Upper Bound (SUUP).
- The dual feasible solution where the value of $\pi(\ell)$ is updated whenever any new request is admitted. Note that when request ℓ' is admitted, the value of $\delta(i,r,t)$ for all locations i where request ℓ' is allocated increases for all time periods t in which ℓ' is active, for all resources r. We call this bound Multi-Update Upper Bound (MUUP).

We consider a cloud system which has 20 data center locations each with its x and y coordinates on an Euclidean plane (just for simplicity). The available amount of resource(s) at each time slot in each location is $U[20:100]$, where $U[l:u]$ denote uniform distribution between l and u. The number of VMs in each request is $U[1:6]$. The number of time periods for which the VM is requested is $U[5:15]$. We assume that the profit is one unit per VM per unit time. In all the experiments we considered the case where all VMs belong to the same class. We consider three sets of experiment: In the first set of experiments, we consider the case where each request is one group of VM and the request specifies the set of locations that are feasible for this request. We call this the *Single Group Case* (SG1). There is one resource and the simulation was over 3000 time slots. In the second and third set of experiments, we assume that each request is for two groups of VMs. The first group of VMs have to be located in a given set of locations, the second group has to be located in another set of locations and the inter-VM group distance is upper bounded. The distance between different locations is assumed to be the Euclidean distance. We can easily extend this approach to the case where there is a network connecting the different locations and the distance between two locations is the "network distance" between the locations. We call this the *Multiple Group Case*. Request ℓ comprises of two groups ℓ_1 and ℓ_2. In the second set of experiments we assumed that there is only one resource and in the third set, we assumed that there are two resources. The third set of experiments were run over 300 time slots while the second set was run over 3000 time slots. Each VM requires $U[1:3]$ units of resource 1 and $U[2:6]$ units of resource 2. The set of experiments with multiple groups with a single resource will be referred to as MG1 and the one with two resources will be referred to as MG2 in the plots.

For the single group case, it is easy to determine the minimum weight mapping. For the multiple group case, the minimum weight mapping is solved as a shortest path problem.

A. Results

In the plots of Figure 3 the x-axis is the experiment number and the y-axis is the solution value and the corresponding upper bounds. The solution value is the total profit accrued by the online algorithm and the upper bounds are the maximum profit that can be achieved by using an offline algorithm that knows all the demands ahead of time. In each plot we compute ratio of all the bounds to the profit. This is done since the main objective of the experiments is to test the actual performance of the algorithm compared to the dual upper bounds. In order to represent all the results on the same scale, we perform this normalization. The dotted line provides the $\frac{e}{e-1}$ upper bound. Note that the computed upper bound SUUB and MUUB are typically tighter than the theoretical upper bound. Note that if all demands are accepted, then the online solution is optimal. The load on the system was set so that some demands will be rejected due to unavailable capacity. In most cases, the computed online solution was within a factor of $1.2-1.3$ of the offline optimal solution.



(a) Single Group-Single Resource (b) Multi Group-Single Resource (c) Multi Group-Two Resources

Fig. 3. Experimental results with different types resource requests.

B. Case study

In order to further evaluate the placement algorithm in a more realistic cloud environment, we have conducted a case study based on the network topology of a typical service provider (see Figure 4), along with more detailed cloud resource configurations. The network consists of 46 nodes, among which 22 nodes with higher connectivity are selected to assign data centers. Each data center has 5,000 physical hosts. Each host has 16 cores, 256GB memory, and 5TB storage. We assume each demand consists of a set of VMs, each with 4 cores, 8GB memory, and 100 GB storage. Number of VMs in each demand is uniformly distributed between 2 and 50. Demand holding time is selected from an exponential distribution with mean of 1,000 hours. We assume each demand requires the VMs to be allocated from data centers that satisfy its delay constraint, i.e., within the specified delay bound to its location. Location of the demand is selected randomly from one of the 46 nodes, and the delay bound is selected from a uniform distribution between 5 and 10 seconds. Demand arrival is generated from a Poisson distribution. We control the offered load to the system by varying the mean arrival rate.

We use SUUP algorithm for this study since it does not require rearranging resource allocations for all demands upon each new demand arrival, and hence requires less computations. We compare it with a *Best Available* algorithm where allocation is done by selecting the closest data center with available resources.

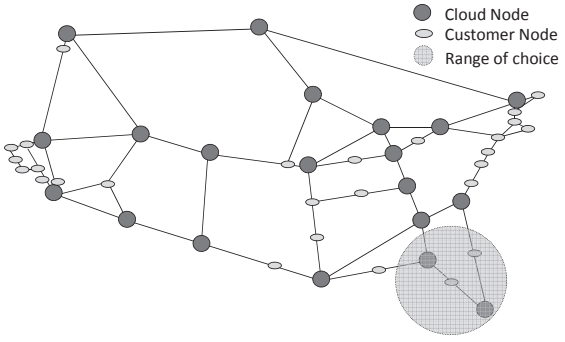


Fig. 4. Network topology

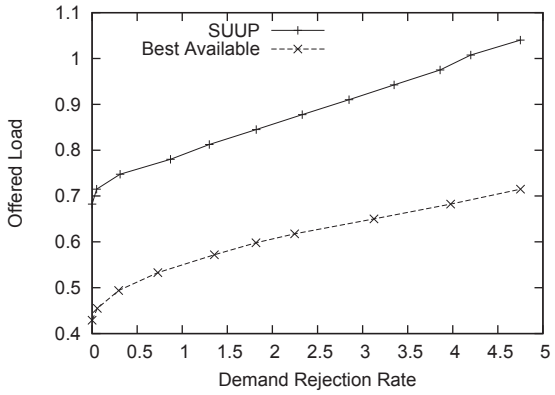


Fig. 5. Offered load under different rejection rate

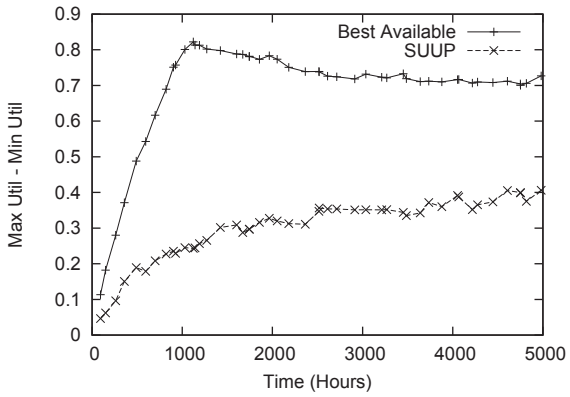


Fig. 6. Variation in resource utilization across data centers

Figure 5 shows the comparison of the two algorithms. It shows the corresponding offered load under a given target demand rejection rate. The offered load is computed as the ratio of the sum of total amount of resource across all demands in all time slots, over the total amount of resource in data centers in all time slots. In this experiment, the offered load is computed based on CPU cores, which is the resource bottleneck. Demand rejection rate is the percentage of rejected demands over all demands. We observe that SUUP can accommodate about 30% more offered load than Best

Available under the same rejection rate. Note that we only study the rejection rate in the range below 5% since we expect the cloud provider will typically try to avoid higher rejection rate.

The reason that SUUP works significantly better than Best Available is that although both algorithms consider the delay constraints of the demands, SUUP optimizes the placement by balancing the load across different data centers. We verify this by running both algorithms under the same offered load (0.68), and sample the CPU resource utilization across all data centers and compute the different between the max and min utilizations. The sample interval is selected from an exponential distribution with mean of 100 hours. Figure 6 shows the difference in max and min resource utilization across data centers at each sampled time point for both algorithms. We observe that the difference is much lower for SUUP, indicating that SUUP leads to much more balanced allocation.

VII. Related Work

There have been some work on placement of VMs within a cloud computing infrastructure. Meng et. al. [10] address the problem of allocating VMs inside a data center to minimize network costs. They present an approximation algorithm for placing "talkative" VMs close to each other in a data center and evaluate the scheme using real-life traffic traces. Alicherry and Lakshman [1] address a similar problem within the context of distributed cloud. They assume that the resource request comes pre-specified with inter-communication among VM pairs, and provide an approximation algorithm for optimal data center selection. Maguluri et. al. [9] investigate the resource allocation in a cloud from the point of view of balancing server loads. They study schemes to optimally fit VMs into the servers. Jiang et. al. [6] study the problem of VM placement in conjunction with the traffic routing among the VMs. They propose a linear cost model for VM allocation and network traffic and propose an online solution for it. We propose a generalized methodology for online resource allocation in a cloud system that can work across different cloud architectures and allocate multiple resources jointly. In addition, it can take into account various constraints on resource placement and can optimize a cloud service provider's specified goal (e.g., revenue).

There is a large body of work that deals with online allocation of virtual circuits in networks [2]. These are online algorithms for routing permanent virtual connections (PVC) in ATM networks. Our approach to the online optimization problem follows the primal-dual approach in [4]. However, in their problem, they assume that once a connection request is accepted, it remains in the system forever. We assume that each resource request has an arrival and departure time. Moreover, we also consider the multi-resource online allocation problem. The idea of modifying the dual update rule in the case where the minimum configuration problem can be solved using an α-approximation algorithm follows the approach in [8].

VIII. Conclusion

In this paper we propose a generalized methodology for online resource placement in a cloud system. We show how different resources (e.g., CPU, memory, disk, etc.) can be jointly allocated to successfully instantiate a requested VM. Our methodology is general enough to handle resource allocation in both traditional cloud and the emerging distributed cloud architectures. We permit the users to specify different types of constraints on resource placement for proper execution of their jobs. We show how such allocation problems can be solved algorithmically, and derive worst case competitive ratios for them. Using experiments we show that our algorithms perform far superior to the worst case in all practical situations.

References

[1] M. Alicherry, and T.V. Lakshman, "Network Aware Resource Allocation in Distributed Clouds", INFOCOM, 2012.

[2] J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts, "On-line load balancing with applications to machine scheduling and virtual circuit routing" . *STOC* , pp. 623-631 (1993).

[3] N. Buchbinder, J. Naor, "Online Primal-Dual Algorithms for Covering and Packing Problems", 13th Annual European symposium on Algorithms, (2005).

[4] N. Buchbinder, J. Naor, "A Primal-Dual Approach to Online Routing and Packing", 47th IEEE Foundations on Computer Science, (2006).

[5] D. Hochbaum, A. Pathria, "Analysis of the Greedy Approach in Problems of Maximum k-Coverage", Naval Research Logistics Quarterly, Vol.45(6), pp.615-627 (1998).

[6] J.W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM Placement and Routing for Data Center Traffic Engineering", INFOCOM Mini-Conference, 2012.

[7] R.M. Karp, U. Vazirani, V. Vazirani,"An Optimal Algorithm for Online Bipartite Matching", 22nd ACM Symposium on Theory of Computing, pp.352-358. (1990)

[8] M. Kodialam, T.V. Lakshman, S. Mukherjee, and L. Wang, "Online Scheduling of Targeted Advertisements for IPTV", IEEE Transactions on Networking, Vol. 19, No. 6, Dec 2011.

[9] S.T. Maguluri, R. Srikant, and L. Ying, "Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters", INFOCOM 2012.

[10] X.Meng, V.Pappas, L.Zhang,"Improving the Scalability of Data Center Networks with Traffic-Aware VM Placement", INFOCOM, 2010.

[11] A. Mehta, A.Saberi, U.Vazirani, V. Vazirani, "Adwords and Generalized Online Matching", 46th IEEE Foundations on Computer Science, pp.264-273, (2005).

[12] SCOPE Alliance, "Telecom grade cloud computing", scope-alliance.org, 2011.