A Stochastic Approach for Virtual Machine Placement in Volunteer Cloud Federations

Abdelmounaam Rezgui  
Department of Computer Science & Engineering  
New Mexico Tech  
801 Leroy Pl., Socorro, New Mexico, 87801, USA  
Email: rezgui@cs.nmt.edu

Sami Rezgui  
Université d’Alger III  
2 Rue Ahmed Ouaked, Dely Brahim  
Algiers, Algeria  
Email: rezguisami@gmail.com

Abstract—Volunteer cloud federations (VCFs) are cloud federations where clouds may join and leave a federation without restrictions and may contribute resources to the federation without long term commitment. This makes it difficult to predict the long term availability of resources. Also, in IaaS VCFs, volunteers may collectively contribute a large number of heterogeneous virtual machine instances. In this paper, we focus on the problem of efficiently allocating this dynamic, heterogeneous capacity to a flow of incoming VM instantiation requests. We propose an approach, called stochastic least differential capacity (SLDC), that allows over-provisioning only when necessary. The approach uses historical information about recent instantiation requests to derive stochastic predictions regarding future demand. We implemented VCFSim, a VCF simulator that uses the proposed resource allocation solution. The results of the experimental evaluation show that the proposed approach is able to improve the success rate of VM instantiation requests by up to 38% compared to an approach that uses exact matching with no demand forecasting.

I. INTRODUCTION

A cloud federation is a loosely coupled, possibly interoperable, collection of clouds that provides to its users the abstraction of a single cloud. Through this abstraction, users see a single pool of services aggregated from different providers. Cloud federations are viewed as a means to address the economic problems of vendor lock-in and provider integration. In addition, they are an alternative to reduce cost (e.g., through partial outsourcing to more cost-efficient regions) and improve performance and disaster-recovery through methods such as co-location and geographic distribution [1].

As the idea of cloud federations matures, scientists are increasingly recognizing that sharing resources from several clouds could provide the computing/storage resources needed to solve many complex science problems. The owners of many private clouds now realize that it is in their mutual benefits to volunteer resources from their individual clouds to form a larger volunteer cloud federation (VCF). This is a more cost effective alternative (than deploying larger clouds) to enable the execution of applications that may not be possible using resources on a single cloud. Volunteer cloud federations enable volunteer cloud computing (VCC), a computing model that combines underutilized computing and/or storage resources on several clouds to quickly solve compute- and/or data-intensive problems.

While volunteer cloud computing is a promising computing model, substantial research is needed to explore architectures, systems, and techniques enabling this model. In this paper, we focus on a fundamental and complex challenge, namely, developing a virtual machine placement technique for VCFs. Traditionally, VM placement has been studied in the context of a single cloud. In a single cloud, efficient VM placement techniques are important because they help achieve one or more objectives including increasing resource utilization, improving load balancing, reducing the need for VM migration, reducing energy consumption, improving fault tolerance, etc. In VCFs, the problem is to determine an efficient mapping between VMs and individual member clouds. Both inter- and intra-cloud VM placement techniques are needed in cloud federations: once a VM placement algorithm for federated clouds determines in which member cloud a VM must be hosted, another algorithm must determine which host of the selected cloud must run the VM.

Solutions to the problem of VM placement in VCFs must take into account that, in VCFs, resources are volunteered and that the owner of a volunteered resource may withdraw it from the federation at any time. Also, users access resources on a VCF with no monetary obligations. VM placement in this case must attempt to maximize the utilization of these resources (not profit) while, possibly, enforcing additional constraints such as fairness or proportional service, i.e., users who also volunteer resources must be able to get a service from the federation that is proportional to the resources that they contribute to the cloud federation.

Compared to the case of single clouds, the problem of VM placement in VCFs remains largely unexplored. In this paper, we focus on the problem of maximizing the number of virtual machines that may be placed on a VCF. Specifically, we propose a new approach for VM placement in VCFs based on the idea of demand forecasting. In this context, the demand is the aggregate number of VM instantiation requests. The proposed approach uses historical information about recent instantiation requests to derive stochastic predictions regarding future demand. Unlike existing approaches for traditional cloud federations, our approach takes into account the unique characteristics of VCFs. In particular, since resources are volunteered (i.e., donated for free), the optimization goal in VCFs is to maximize resource utilization, i.e., maximize the number of virtual machines admitted to the federation. The proposed solution uses a model of VCFs where member clouds expose their capacity in terms of number and size of computing...
instances (i.e., VMs) that they are willing to contribute to the cloud federation.

The remainder of this paper is organized as follows. In the next section, we present our model of cloud federations. In Sections III and IV, we present the theoretical details of the proposed approach for VM placement in VCFs. In Section V, we present an experimental evaluation of the proposed solution. Finally, in Section VI, we give concluding remarks and some future research directions.

II. Cloud Federation Model

We consider a cloud federation (Figure 1) where each cloud owner contributes a certain amount of resources. When joining the federation, a cloud owner contributes resources to the cloud federation in the form of a set of VM instances of different sizes. When a new set of instances is made available by a contributor, the cloud federation controller updates its list of available instances accordingly. In practice, private clouds may be built using different software packages. As a result, different federation members may define heterogeneous VM instances, i.e., instances that differ in terms of resources such as CPU speed, disk or memory size. Let \( R \) be the list of \( m \) resources over which instances are defined. An instance type \( IT_i \) is defined as a tuple \( (r_{i1}, r_{i2}, ..., r_{im}) \) where \( r_{ik} \) is the amount of resource \( k \) offered in instances of type \( IT_i \).

An instance of type \( IT_1 \) would have 2 CPU cores of 2 Ghz each, 2 GB of memory, 150 GB of disk, and a network bandwidth of 100 MB/s. A volunteer may, for example, contribute ten instances of type \( IT_1 \) and five instances of type \( IT_2 \). We define a contribution as a set of pairs \( \{(IT_1, c_1), ..., (IT_n, c_n)\} \)

where \( c_i \) is the number of instances of type \( IT_i \) volunteered by a given cloud owner.

Definition 1:

Let \( D \) be the set of member clouds in the federation. At any given time, the current capacity of the federation is the union of all contributions made by all its member clouds and not currently allocated to any user.

Definition 2:

Let \( IT_i \equiv (r_{i1}, r_{i2}, ..., r_{im}) \) and \( IT_j \equiv (r_{j1}, r_{j2}, ..., r_{jm}) \) be two instance types. \( IT_i \) is said to dominate \( IT_j \) if and only if:

\[
\forall k, 1 \leq k \leq m : r_{ik} \geq r_{jk}
\]

We note this by: \( IT_i \in Dom(IT_j) \). Note that the relationship dominate is a partial order over all instance types of the cloud federation.

Users of the federation submit their requests in terms of VM instantiation requests (Figure 1). Each request specifies an instance type \( IT_i \). The federation controller then attempts to determine a cloud member that still has capacity to create such an instance. Two cases may occur:

Case 1: if a member cloud is found that has capacity for, at least, one instance of the requested type, the federation manager forwards the user’s request to that member cloud. In general, if the requested instance type is available on several member clouds, the federation controller chooses one of the member clouds based on some policy such as: cloud geographically closest to the user, round-robin, cloud with highest remaining capacity first, etc.

Case 2: if no member cloud has available instances of the requested type, the federation controller may either reject the request or create an instance larger than (i.e., dominating)
the one requested (if, of course, one can be found). If the latter approach is adopted, a large number of options (i.e., combinations of member clouds and their hosted instances) may satisfy the submitted request. A significant challenge is to determine an efficient mapping that determines which member cloud to host the new VM and which instance in this member cloud is the most suitable for the new VM. In this context, the term “efficient” means “in a way that leaves as much capacity as possible available for future VM instantiation requests.”

III. LEAST DIFFERENTIAL CAPACITY APPROACH

To create a virtual machine on the cloud federation, a user first selects an instance type $IT_i$ that is appropriate to his/her application(s) and submits a request to the federation controller to create a VM of the desired type. The federation controller then uses a VM placement approach to determine the best member cloud to host the new VM and which instance in this member cloud is most suitable for the new VM. If an exact match is not available, the federation controller must select a VM instance among a large number of dominating VM instances that may be available on several member clouds. An easy alternative in this case is to randomly allocate one of those larger-than-requested VMs to the user’s request. This VM over-provisioning, however, can quickly exhaust resources on the member clouds. In this section, we introduce a basic scheme, called the Least Differential Capacity (LDC) approach that attempts to create VMs while minimizing the “wasted” capacity, i.e., capacity in excess to what the user is requesting.

It is important to note that the problem addressed in this paper cannot be addressed by simply adjusting the size of VM instances available on the volunteered clouds to match the size of the VM requested by users. With current virtualization technology, the overhead of VM resizing has become negligible. However, in the context of VCFs, cloud owners generally do not allow external entities (e.g., a cloud federation’s controller) to make changes that may impact their clouds, e.g., changing the volunteered capacity. We first present the intuition through an example.

Consider a scenario (Figure 2) where $m$, the number of resources is 2; disk and memory, and instance types are defined as tuples $(r_1, r_2)$ where $r_1$ and $r_2$ are the disk and memory sizes of the given instance type. Assume that, at a given time, the cloud federation’s available capacity consists of four instances of types $IT_1$, $IT_2$, $IT_3$, and $IT_4$. Figure 2 is a graphic representation of the cloud federation’s instance types in a 2D plane. First, a VM creation request is received for type $IT_1$. The federation has capacity to create one VM of type $IT_1$. It creates a VM of type $IT_1$. Assume that the next request is also the creation of a VM of type $IT_1$. Since there is no available instance of type $IT_1$, the LDC approach will select instance type $IT_2$ to satisfy the current request since $IT_2$ is the closest in terms of capacity to $IT_1$ compared to $IT_3$ and $IT_4$. The “wasted” capacity corresponds to the rectangle labeled 1 in Figure 2. After this operation, the cloud federation has no more available instances of type $IT_2$. Assume that the following VM creation request is of type $IT_2$. Since this request cannot be satisfied as requested, the federation controller will create a larger instance of type $IT_3$ to satisfy the current request. Here again, there will be a “wasted” capacity that corresponds to the rectangle labeled 2 in Figure 2. Notice that the instance $IT_4$ cannot be used to satisfy this third request because $IT_4 \notin Dom(IT_2)$.

Now, consider a scenario where the cloud federation is able to probabilistically predict the sequence of future VM creation requests. In the previous example, the sequence of requested instance types was: $IT_1$, $IT_1$, and $IT_2$. In this new scenario, the federation controller will process the first request as it did before, i.e., create a VM of type $IT_1$. When it receives the second request, the federation controller may exploit the probabilistic knowledge that a request of type $IT_2$ is imminent and, as a result, spare the remaining available instance of type $IT_2$ for that likely future request and create a VM of type $IT_4$. Notice that this is possible since $IT_4 \in Dom(IT_1)$. When the third request of type $IT_3$ is received, the federation controller will be able to create a VM of the exact requested size without over-provisioning. In this scenario, the “wasted” capacity corresponds to the rectangle labeled 3 in Figure 3. As it is shown in Figures 2 and 3, the total “wasted” capacity in the second scenario (rectangle 3) is much smaller than the total “wasted” capacity in the first scenario (sum of rectangles 1 and 2). The basic idea here is to exploit probabilistic information about the likelihood that the instantiation of a VM of a certain size will be requested in the near future. The example shows that the basic LDC approach does improve the success rate when processing VM instantiation requests. However, it also shows that it may yield sub-optimal VM allocations resulting in substantial capacity loss. We now elaborate on the proposed stochastic least differential capacity approach.

IV. STOCHASTIC LEAST DIFFERENTIAL CAPACITY APPROACH

In the previous section, we showed the limitation of the LDC approach. To address this limitation, we introduce the stochastic least differential capacity (SLDC) approach which improves on the basic LDC approach by exploiting two elements, namely: (i) the current VM supply, i.e., current available capacity for each instance type and (ii) a stochastic information about future demand, i.e., a probabilistic knowledge (derived by extrapolating recent history) about the types of future VM instantiation requests.

We now associate a pair $(c_i, \mu_i)$ with each instance type $IT_i$. $c_i$ is the current available capacity for instances of type $IT_i$, i.e., the total count of instances of type $IT_i$ available on all member clouds. $\mu_i$ is the probability that, in the near future, the federation receives a VM instantiation request of type $IT_i$. Several approaches may be used to evaluate $\mu_i$. We present our approach to evaluate $\mu_i$ in the next section.

Consider a cloud federation that offers $n$ instance types of VMs. We represent VM instance types as points in an $m$-dimensional space where each dimension corresponds to a resource (see Figure 2). In this space, VM instance type $IT_i$ corresponds to the point whose coordinates are $(r_1^i, r_2^i, \ldots, r_m^i)$. We introduce a matrix $\Delta$ of size $n \times n$, called the pairwise capacity distance matrix. Intuitively, for two instance types $IT_i$ and $IT_j$, $\Delta(i,j)$ is a measure of the difference in capacity between $IT_i$ and $IT_j$. Formally, $\Delta$ is defined as follows:

$$\Delta(i,j) = \begin{cases} \| IT_i, IT_j \|, & \text{if } IT_j \in Dom(IT_i) \\ +\infty, & \text{otherwise} \end{cases}$$  

(1)
Initialization

\[ h = 0; \]

/* \( h \) is the index in the history \( \mathcal{H}_q \) */

When a request to create a VM of instance type \( IT_j \) is received

if an instance \( I \) of type \( IT_j \) is available

allocate the instance \( I \) to the request

else

if necessary

Update \( DIC(i,*) \) the vector of differential instantiation costs relevant to instance type \( IT_i \);

/* In practice, the total number of instance types is small. The cost of this update is therefore in the order of \( O(1) \) */

endif

Find a VM instance type \( IT_j \) such that:

\( IT_j \in Dom(IT_i) \) and \( DIC(i,j) = \min_k DIC(i,k) \)

/* Again, as the total number of instance types is small in practice, the cost of this operation is in the order of \( O(1) \) */

if found

Create a VM instance using instance type \( IT_j \);

endif

\[ \mathcal{H}_q[h] := i \]

\[ h = (h + 1) \% q; \]

\[ c_i := c_i - 1; \]

endif

Fig. 4: Stochastic Least Differential Capacity (SLDC) Algorithm

Where \( \| IT_i IT_j \| \) is the length of the vector \( IT_i IT_j \), i.e.,

\( \| IT_i IT_j \| = \sqrt{\sum_{k=1}^{m}(r_{i,k}^j - r_{i,k}^i)^2} \). This definition assumes that all resources have equal importance. In practice, to better capture the distance between VM instance types, cloud providers may associate different weights to different resources, if \( \alpha_k \) is the weight of resource \( r_{i,k} \), \( \Delta(i,j) \) may be given as:

\[
\Delta(i,j) = \begin{cases} 
\sqrt{\sum_{k=1}^{m}\alpha_k (r_{i,k}^j - r_{i,k}^i)^2} & \text{if } IT_j \in Dom(IT_i) \\
+\infty & \text{otherwise} 
\end{cases}
\] (2)

Normally, the matrix \( \Delta \) may be computed entirely offline. It must be updated only when new instance types are made available to the federation or when the federation ceases to offer a given VM instance type. For example, when a new member joins the federation and volunteers new VM instance types, \( n \) (\( \Delta \)'s dimension) must be increased to accommodate the newly introduced VM instance types. Also, when the size of an existing VM instance type changes, \( \Delta \) must be updated to reflect this change.

Given a request to create a VM of type \( IT_i \), if an exact match is not found, the SLDC algorithm uses over-provisioning to satisfy this request. Let \( IT_j \) be the VM instance type allocated to the user’s request instead of the requested instance type \( IT_i \). To measure the capacity “waste” resulting from this over-provisioning, we introduce the concept of differential instantiation cost (DIC) defined as follows for a given pair of instance types \( IT_i \) and \( IT_j \) where: \( IT_i \) is the requested instance type and \( IT_j \) is an instance type such that: \( IT_j \in Dom(IT_i) \):

\[
DIC(i,j) = \frac{\mu_j}{c_j} \Delta(i,j)
\] (3)

Intuitively, \( DIC(i,j) \) is the cost of creating a larger instance of type \( IT_j \) when the the actual request specifies a smaller instance of type \( IT_i \). The formula captures three elements:

\( \mu_j \): the instantiation probability of type \( IT_j \). The cost is obviously higher if it is more likely that, in the future, more requests of type \( IT_j \) will be received.

\( c_i \): the current count of instances of type \( IT_i \). The cost is clearly inversely proportional to the “abundance” of instances of type \( IT_i \). Allocating an instance of type \( IT_j \) to the current request makes instances of type \( IT_j \) scarcer which, in turn, results in an increase in the cost of creating instances of this type.

\( \Delta(i,j) \): this value captures how much excess capacity we are wasting by allocating an instance of type \( IT_j \) instead of the smaller requested VM of type \( IT_i \).

We now turn to the issue of forecasting future instantiation probabilities (i.e., \( \mu_j \) in Equation 3). Several approaches may
be adopted to predict VM instantiation probability. We adopt a history-based approach where the probability of an instance type to be requested in the future is derived from the history of past VM creation requests that the cloud federation has received up to the current time. The federation controller maintains an instantiation history, noted $\mathcal{H}$, of VM creation requests. $\mathcal{H}$ is a list where the $i^{th}$ entry $\mathcal{H}[i]$ is the $i^{th}$ instance type requested. Taking into account the entire VM instantiation history may not be accurate in practice. For example, many users may have created VM instances far in the past, e.g., several months ago. Including these old requests when evaluating VM instantiation probabilities may distort the result. To improve accuracy, we limit the computation to only the most recent history, i.e., the most recent $q$ requests. This subset of $\mathcal{H}$ is noted $\mathcal{H}_q$. Let $v_i$ be the number of times the cloud federation receives a VM creation of type $IT_i$ in the partial history $\mathcal{H}_q$. A simple formula to determine $\mu_i$ at the current time may be: $\mu_i = \frac{V_i}{\| \mathcal{H}_q \|} = \frac{v_i}{q}$. This formula, however, gives equal weights to old and recent requests. We argue that recent history tends to repeat itself and that recent history is more relevant than older history. In practice, this tends to be true. For example, for a few weeks, the first author and other co-authors were working on these two scientific problems: [3] and [2]. During that period, they frequently used a number of similar Amazon EC2 instances. If a cloud federation were used instead, the federation would have been processing similar requests for a certain period of time. To exploit the information about recent history, we define a weight function $w$ over VM instantiation events in the history $\mathcal{H}_q$. $w$ is a function that increases linearly from $1/q$, which is the weight associated with the oldest VM instantiation event (i.e., the first in the current moving window) to $q/q = 1$, which is the weight associated with the most recent VM instantiation event in the partial history $\mathcal{H}_q$ (i.e., the $q^{th}$ in the current moving window). We note this by: $w_j = \frac{j}{q}, \forall j, 1 \leq j \leq q$.

To normalize (since the objective is to determine probabilities), we first compute $W$, the sum of weights for all instantiation events: $W = \sum_{j=1}^{q} w_j = \sum_{j=1}^{q} \frac{j}{q} = \frac{q+1}{2}$. $\mu_i$ can now be computed as:

$$\mu_i = \frac{\sum_{j=1}^{q} \mathcal{H}_q[j]=i w_j}{W} = \frac{2}{q(q+1)} \sum_{j=1}^{q} \mathcal{H}_q[j]=i$$

In terms of complexity, computing $\mu_i$ as given in the previous equation is clearly in the order of $\Theta(\| \mathcal{H}_q \|) = \Theta(q)$. In practice, however, it is easy to use data structures that lower this complexity to $O(q)$.

Figure 4 gives the details of the SLDC algorithm.

V. EXPERIMENTAL EVALUATION

We implemented VCFSim, a simulator of VCFs based on our federation model presented in Section II. VCFSim uses the SLDC approach for VM instantiation. To determine the types of the volunteered VM instances, we adopt the characteristics of Amazon EC2 instances shown in Table I. The simulator first builds the federation by having a number of member clouds join the federation. We focused on a scenario of a static cloud federation, i.e., members do not leave the federation. To add a cloud to the federation, VCFSim generates a (different) random volunteered capacity for each member cloud. VCFSim also randomly distributes this capacity, i.e., it randomly selects how many instances of each type the new member cloud is willing to contribute.

To evaluate the SLDC algorithm, we focused on two aspects: (i) the absolute impact of the SLDC algorithm on increasing the federation’s usable capacity and (ii) the relative impact of the SLDC algorithm compared to scenarios that use random over-provisioning. Figure 5 shows how the SLDC algorithm contributes to improving the success rate of VM instantiation requests. The figure shows the results of 20 simulations. Each simulation first forms a VCF having the configuration Config 1 of Table II. In Figure 5, the x-axis corresponds to the (randomly generated) federation’s capacity. The solid line corresponds to the number of successful VM allocations without applying the SLDC algorithm. The dotted line corresponds to the additional VM allocations made possible by applying the SLDC technique. The experiment shows that, in this scenario, the proposed technique improves the success rate by a factor that ranges from 12% to 38%. The average improvement was 23%.

TABLE II: Configurations Used in the Experiments

<table>
<thead>
<tr>
<th></th>
<th>Config 1</th>
<th>Config 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Member Clouds</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Number of VM Types</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Max Number of Volunteered VMs</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>History Size</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Fig. 5: Effect of SLDC on Capacity (Config 1)

Fig. 6: Effect of SLDC on Capacity (Config 2)

---

1http://aws.amazon.com/ec2/instance-types
We ran the same 20 simulations with a larger federation (configuration Config 2 in Table II). Figure 6 shows the effect of the SLDC technique in this larger configuration. The experiment shows that, in this case, the proposed technique improves the success rate of VM instantiation requests by a factor that ranges from 10% to 17%.

In the second set of experiments, we studied the relative impact of the SLDC algorithm compared to scenarios that use random over-provisioning. Figure 7 shows the results of over 20 simulations. As in the case using SLDC, each simulation first forms a VCF having the configuration Config 1 of Table II. The experiment shows that, in this configuration, random over-provisioning improves capacity utilization by a factor that ranges from 9% to 35%. The average improvement was 20%. Compared with the SLDC scenario (where the average is 23%), the difference in this small configuration is not consequential. We also ran the same set of experiments (random over-provisioning) using Config 2 of Table II. The results (Figure 8) show that, in this case, random over-provisioning improves the success rate of VM instantiation requests by a factor that ranges only from 5% to 11%.

The experiments discussed in this section show two results. First, over-provisioning (whether random or SLDC-based) can improve the success rate of VM instantiation. The improvement is much more pronounced in the case of SLDC-based over-provisioning. Second, in both types of over-provisioning, the improvement tends to be more substantial in small/medium cloud federations than in larger federations. This finding can be explained by the fact that, as a cloud federation gets larger, capacity also increases. As a result, VM instantiation requests will then increasingly be satisfied without the frequent need for over-provisioning.

VI. CONCLUSION

We presented a probabilistic approach to the problem of virtual machine placement in volunteer cloud federations. The solution, called the stochastic least differential capacity (SLDC) approach, is based on three elements: (i) evaluating the probability of future demand in terms of virtual machine instantiation requests using recent history, (ii) taking into account the current supply in terms of VM availability when making VM allocation decisions, and (iii) minimizing the capacity waste when exact virtual machine matching is not possible. To evaluate the proposed technique, we implemented VCFSim, a simulator for VCFs. Our experiments showed that the SLDC approach can potentially increase the success rate of VM placement by up to 38% compared to traditional approaches.

The solution presented in this paper forecasts future VM demand but does not forecast future VM supply (it only uses current information about VM supply). We plan on improving this solution by forecasting both supply and demand. We also plan on extending the current approach to dynamic VCFs where member clouds may join and leave the federation and may also change their resource contribution to the federation.

ACKNOWLEDGMENT

This research is supported in part by a grant from the NM NASA EPSCoR.

REFERENCES

