End-to-End Informed VM Selection in Compute Clouds

Mario Meireles Teixeira
Department of Informatics
Federal University of Maranhão
São Luís, MA, Brazil
Email: mario@deinf.ufma.br

Azer Bestavros
Computer Science Department
Boston University
Boston, MA, USA
Email: best@bu.edu

***Abstract*—The selection of resources, particularly VMs, in current public IaaS clouds is usually done in a blind fashion, as cloud users do not have much information about resource consumption by co-tenant third-party tasks. In particular, communication patterns can play a significant part in cloud application performance and responsiveness, specially in the case of novel latency-sensitive applications, increasingly common in today's clouds. Thus, herein we propose an end-to-end approach to the VM allocation problem using policies based uniquely on round-trip time measurements between VMs. Those become part of a user-level 'Recommender Service' that receives VM allocation requests with certain network-related demands and matches them to a suitable subset of VMs available to the user within the cloud. We propose and implement end-to-end algorithms for VM selection that cover desirable profiles of communications between VMs in distributed applications in a cloud setting, such as profiles with prevailing pair-wise, hub-and-spokes, or clustered communication patterns between constituent VMs. We quantify the expected benefits from deploying our Recommender Service by comparing our informed VM allocation approaches to conventional, random allocation methods, based on real measurements of latencies between Amazon EC2 instances. We also show that our approach is completely independent from cloud architecture details, is adaptable to different types of applications and workloads, and is lightweight and transparent to cloud providers.**

I. Introduction

In current cloud utilization scenarios, a customer requests and obtains from the cloud provider a set of virtual machines (VMs) with generic processing capabilities that should be organized in such a way to provide the best possible services with the least consumption of resources and hence the least cost to the customer. The allocation of generic compute and network resources to customer workloads so as to provide the best performance at the least cost is a classical resource management problem with many variants, dating back to the management policies of mainframe computers. Today, in Infrastructure as a Service (IaaS) clouds, the problem to be tackled is not very much different, although some complicating issues arise, such as application heterogeneity, geographically dispersed nodes, multiple administrative domains and so on.

When cloud customers request a set of VMs from a cloud provider such as Amazon EC2, they usually do not have any information about which physical machine each VM is actually residing on and neither about how these shared physical resources are used by co-tenants, in terms of computational, data and network consumption. Moreover, customers do not know how the VMs in their allocated set relate to one another in terms of delay inside the cloud's network. Although this 'opaque' cloud computing is necessary and even welcome in a scenario with so many independent customers, most cloud customers would certainly benefit from a more informed application-to-VM mapping that would speed up the completion of tasks (and correspondingly reduce cost) and provide a more clever utilization of contracted resources.

In this context, we propose a *Recommender Service* to be used by cloud costumers who wish to make better use of the cloud resources they contracted, by making more informed VM allocation decisions that neither waste nor over-utilize those resources. For instance, consider a situation in which one wishes to run an application within the cloud that requires the availability of k VM clusters, each of m VMs, distant from one another by network latency less than msec. In current cloud settings, selecting VMs that satisfy such a requirement is very difficult if not impossible, at least from a customer's viewpoint, since the environment surrounding the allocated VMs (latency included) is completely hidden.

This is precisely the situation where our Recommender Service will come into play: given the characteristics of a customer's distributed application, this service will be able to identify the subset of VMs that best suit these characteristics at any particular point in time. These VMs would be identified from a larger set of VMs allocated by the cloud provider to the customer, or to an aggregator or broker working on behalf of a number of customers [1].

Our main focus here is not on VM placement from the perspective of the cloud administrator, as this problem has been well studied recently [2]–[4]. Instead, we are particularly interested in empowering the cloud customers' perspective who have been using cloud resources up to now as a black box, in the hope that cloud providers know what is 'best' for them.

An important motivation for our work is that cloud infrastructures have evolved from smaller datacenters, where computing resources are relatively close to one another, to the current settings where VMs are scattered inside enormous datacenter infrastructures or even in multiple datacenters, in geographically distant locations. At the same time, cloud applications are no longer uniquely of the compute intensive or data intensive type; new types of application have emerged, such as gaming, real-time, multimedia and sensor applications, for which latency issues are of paramount importance. Additionally, recent research has shown that intercommunication patterns among VMs can play an important and decisive role in application efficiency [5] and definitely have an impact on application performance and responsiveness as perceived by end users.

Therefore, efficiently using the network inside the cloud is critical, and not only for those delay-sensitive applications. In general, communication latency issues can delay job completion times within the cloud [3] and consequently hinder the quality of services provided. Therefore, in this study we settle for an end-to-end approach for VM allocation inside IaaS clouds, which we believe will result in a better usage of cloud resources by their customers.

More specifically, in this paper we consider VM allocation algorithms that employ round-trip times (RTT) between VMs as a heuristic to support VM allocation decisions. Real-time inter-VM latency measurements serve as input to the Recommender Service, responsible for selecting a subset of m VMs out of n possibilities, subjected to certain requirements set by the cloud customer. The output of the Recommender Service will be a VM mapping that can even follow certain predefined topologies, such as a hive, a star or a set of VM pairs.

The proposed algorithms are implemented and validated through simulation. They are employed to find pairs of VMs with minimal latency between them, choose a machine as the center of a subset of VMs, and to identify VM candidates that may function as local hubs within the cloud. Those informed selections are compared to random, conventional approaches and their results prove to be superior in all cases.

The end-to-end VM allocation strategies described herein are topology agnostic, i.e., they are deliberately blind regarding the cloud's internal architecture and base their allocation decisions exclusively on round-trip time measurements between VMs. This is an innovation and is in contrast to similar research [2], [3], which usually employs some knowledge of the underlying datacenter architecture for that purpose. We argue that this approach is consistent with emerging models of cloud and inter-cloud marketplaces, such as in the *Massachusetts Open Cloud* (MOC) initiative [6].

This paper is organized as follows: Section 2 discusses the current cloud utilization scenario and outlines the Recommender Service. Section 3 highlights the VM selection policies and algorithms proposed herein and their respective applicability. In Section 4, the algorithms are evaluated through simulation using a reference architecture validated with real Amazon EC2 measurements, and the main results are discussed. Section 5 comments on related research and, finally, Section 6 discusses the main conclusions and intended future work.

II. System Architecture

A. Background

Customers of cloud computing infrastructure services (IaaS) typically access a public cloud provider through a web interface and make requests for generic virtual machines which they intend to employ in different types of computing tasks. The underlying idea here is that of computing as a utility

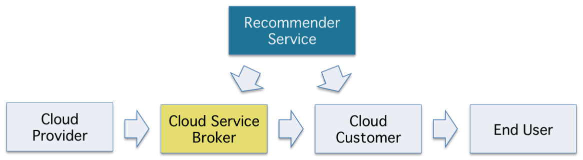

Fig. 1. Emerging cloud services contracting and provisioning model

because the clients do not use a specific cloud application but rather request access to a remote computing platform to be used for any purpose they may think fit. The computing resources requested (usually VMs) may come prepackaged with some software and normally belong to a certain range in terms of computing power — tiny, small, medium and so on, depending on each cloud provider's terminology.

Many providers also make available some kind of monitoring service to customers whereby they can obtain coarse and fine-grained information about how their applications are using their allocated VMs in terms of CPU, data and network consumption. However, these monitoring services do not provide any visibility regarding the impact of third-party applications concurrently running beside their own applications (e.g., in another VM on the same physical machine) or anywhere else in the cloud. The lack of such knowledge can lead to truly unwise application-to-VM mappings.

Our intention here is to provide a service that will collect cloud usage information by one side and couple them with customer requirements on the other side, in order to suggest a VM mapping in accordance with customer needs and suitable to current datacenter resource utilization. Such a service would be especially useful in emerging cloud business models, as shown in Figure 1, where there are brokers that contract a batch of resources from the cloud provider and act as resellers (or aggregators) between the cloud provider and the cloud customer [1], [6], [7].

B. Recommender Service

In this context, we propose a **Recommender Service**, as shown in Figure 2. Cloud customers will make a request for VMs that should meet specific requirements (1), such as a minimum RTT threshold between them or a particular VM topology within the cloud.

The Recommender maintains a set of n VMs previously contracted from a cloud provider and periodically monitors them for performance information (2). As a request arrives, the Recommender will select a subset of m VMs out of the total n that should satisfy the customer's demands. It then suggests this VM arrangement to the requester (3), which in turn deploys applications on the VMs, accordingly. As a result, the customer can choose among the available resources with more certainty, making more informed application-to-VM commitments.

The Recommender does not function as a broker, as it does not allocate the VMs itself. What it actually does is monitor the environment surrounding each VM (e.g., network latency or throughput), in order to be able to better balance its clients' requests over the available resources, providing a VM selection tailored to their needs.

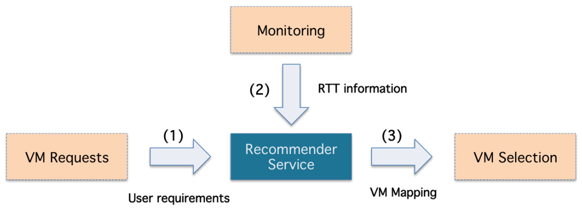

Fig. 2. Architectural components of proposed Recommender Service

We believe such a service can become a centerpiece in cloud resource management in the near future, as applications become more and more complex and with more stringent demands, that cannot be satisfied uniquely by current cloud provider-to-customer models[1].

A fundamental part of the Recommender Service are the VM Selection Algorithms and also the performance information available to them. In the present study, we are particularly interested in algorithms that leverage end-to-end latency information between VMs for allocation decisions.

III. VM Selection Algorithms

This section details the approach for end-to-end latency monitoring between VMs and also discusses four algorithms we propose for VM selection within the cloud, all of which use some sort of delay information for their placement suggestions.

A. Latency Matrix

The primary input for the algorithms is a *Latency Matrix* ($RTT_{i,j}$) that is periodically updated with round-trip time information between any pair (i, j) of VMs known to (or managed by) the Recommender Service:

$$RTT_{i,j} = \begin{bmatrix} 0 & t_{1,2} & t_{1,3} & \dots & t_{1,n} \\ t_{2,1} & 0 & t_{2,3} & \dots & t_{2,n} \\ t_{3,1} & t_{3,2} & 0 & \dots & t_{3,n} \\ \dots & \dots & \dots & \dots & \dots \\ t_{n,1} & t_{n,2} & t_{n,3} & \dots & 0 \end{bmatrix} \quad (1)$$

Each row or column in the matrix represents a VM inside the cloud and the element (i, j) contains the round-trip time between VM$_i$ and VM$_j$. As this information is symmetric, the equivalent element (j, i) is ignored and the values in the main column of the matrix are set to zero. For all implementation purposes, we assume an upper triangular matrix.

We chose to use the latency as the basis for the VM selection algorithms because RTTs are a reliable metric for measuring network distance within clouds, as recent research has pointed out [3], [5], [8]–[10]. Additionally, RTTs are computationally cheap and straightforward to use and obtain. In our study, latency measurements will be employed to estimate the distance between VMs and will constitute a fundamental parameter for VM allocation recommendations.

[1]Note that this sort of service is not necessarily viewed favorably by the providers, as they would rather leave their customers in 'cloudiness' regarding VM selection and thus retain their sole jurisdiction over the cloud's current physical resources utilization.

Nevertheless, recent studies have shown that RTT figures between VMs may vary widely in a cloud environment and this is due mainly to the virtualization overhead since the CPU has a dual role in both computation and networking [9]. Consequently, it is not true that network adjacent VMs in a cloud environment will necessarily be physically close to one another, as is generally the case in network topologies such as that of the Internet. Moreover, latency times may fluctuate over time and thus a particular VM placement valid presently may not be the best option a few moments later. Hence, the latency matrix needs to be updated frequently in order to reflect as timely as possible the status of network distance between the VMs.

To verify how this would work in a real-world set up, we performed a round-trip time monitoring experiment using Amazon EC2 machines. First, we implemented and deployed a client-server PING application in each of the rented VMs. We then made the PING client periodically wake up and send a message to each of the other virtual machines, in a round robin fashion. This step was repeated a number of times (set to 10 in our experiments) and, as the client received the return messages, it computed the average delay for each (VM$_i$, VM$_j$) pair and updated the figures in the Latency Matrix.

To assess the complexity of this algorithm, consider that each VM needs to ping all other VMs except itself. Hence, we have $2(n - 1)$ round-trip messages per VM, giving a total of $2n(n - 1)$ messages for n VMs. Thus the complexity is $O(n^2)$ for each updating cycle. The sheer number of messages exchanged may seem forbidding but remember that the PING messages are short, about 64 bytes each. Additionally, this monitoring applies only to the machines known to the Recommender Service (not to all VMs in the cloud) and it can be done on demand, aiming at the machines of a particular cloud customer currently asking for VM placement recommendations. Our experiments indicated that VM performance was not significantly affected by RTT monitoring.

B. Hive Algorithm

This algorithm aims to find a set of VMs close to one another by a round-trip time less than r ms, i.e., the RTT between any pair of VMs belonging to that set should be less than r. A variation of that would be to mandate that the 95^{th} percentile of the RTT_{ij} measurements be less than the threshold r, in order to avoid that a good arrangement of VMs be discarded because of a few outliers.

We call this approach the *Hive Algorithm* as it intends to find clusters of nearby VMs in an attempt to exploit locality of reference. In fact, most applications in the cloud may benefit from such an approach because it minimizes intra-cloud traffic and as a result lowers bandwidth usage. It also helps reduce datacenter fragmentation since the VMs cooperating in a computation tend to be brought together. However, note that distance in this case is usually but not always related to physical distance. Although VMs on the same blade or rack tend to have lower RTT times between them, in some cases this may not be true due to virtualization or workload issues. And thus perhaps a VM on another rack might be the closest neighbor.

The Hive Algorithm is very difficult to implement in practical terms. It is actually a special case of an NP-hard graph problem. What we are trying to find here is a subgraph wherein we can only draw an edge from any node i to any node j if the RTT between them is less than the specified threshold. The greater the number m of VMs in the hive, the harder the complexity of the algorithm. If we wanted to find the largest possible hive subjected to $RTT_{ij} < distance$ then we would come to the MAXCLIQUE problem [11], where the challenge is to find a clique of maximum size, what is NP-hard.

One way to address this problem is to restrict the number of VMs in the hive to be very small. In fact, in Section III-E we give an example of a simple approach to find m pairs of VMs inside the cloud with an RTT between them lower than a certain threshold.

C. Star Algorithm

Our second approach, the *Star Algorithm* seeks to find the answer to a simple question: given a set of VMs inside the cloud, we would like to know which VM is approximately in the center of them, considering latency. In other words, we want to organize this set of VMs according to a star topology whose center (the selected VM) is closer on average to all its neighbors. This can be mathematically defined as follows:

$$RTT_{k=1}^{n} = \frac{\sum_{i=j=1}^{n} t_{ij}}{n-1}, i \neq j \tag{2}$$

$$VM_{center} = min\{RTT_k\} \tag{3}$$

We present Algorithm 1 which, for a set of VMs, will return the VM with the lowest average of RTTs in relation to all other VMs in the set, i.e., the one to be chosen as the center of the star. In Line 2, we sum all elements (i, j) to the right side of the main diagonal of the matrix and, in Line 3, we sum all elements above it, taking advantage of the symmetry of round-trip times. Thus our matrix can occupy only half of the memory required by $n \times n$ elements. The complexity of this algorithm is clearly $O(n^2)$.

Algorithm 1 Star

```
Input: $RTT$ : Latency Matrix for $n$ VMs
1: for $i = 0$ to # rows in $RTT$ do
2:     $sumx \leftarrow \sum_{j=i+1}^{n} t_{ij}$
3:     $sumy \leftarrow \sum_{j=0}^{i-1} t_{ji}$
4:     $avg \leftarrow (sumx + sumy)/(n-1)$
5:     if $avg$ is the lowest one until now then
6:         $center \leftarrow i$
7:     end if
8: end for
9: return $center$
```

This algorithm can be useful to any distributed application where one node should be in the midpoint with respect to the other nodes in the set. This could be the case of a VM functioning as a name directory, a key distribution service or multicasting a stream of data to the other VMs. In a standard provider-to-customer model, where the latter is simply allotted a number of VMs, it is usually very hard to guess which one of the VMs should be sorted out as the hub. Thus, in all of those cases, the cloud customer could clearly benefit from a more informed allocation of VMs.

D. Best Centers Algorithm

We could generalize the approach of the above Star Algorithm and say we want to choose not just one VM as the center, but rather find the m best VMs that could function as hubs (or super-nodes), aggregating the traffic from neighboring VMs inside the cloud. As example of situations where this is needed, consider an application that requires the maintenance of an overlay network or a distributed hash table with super-nodes used for managing close-by clusters.

We propose Algorithm 2 that will find m centers out of n VMs, all of which have the lowest average RTT to all other VMs in the set, as in the following:

Algorithm 2 BestCenters

```
Input: $RTT$ : Latency Matrix for $n$ VMs; $m$ : number of best
    centers
1: for $i = 0$ to # rows in $RTT$ do
2:     $sumx \leftarrow \sum_{j=i+1}^{n} t_{ij}$
3:     $sumy \leftarrow \sum_{j=0}^{i-1} t_{ji}$
4:     $avg \leftarrow (sumx + sumy)/(n-1)$
5:     $centers\{\} \leftarrow [i, avg]$
6: end for
7: $best\{\} \leftarrow centers\{\}$ sorted by key $avg$ ascending
8: return $best\{0..(m-1)\}$
```

The *Best Centers* algorithm aims to identify the VMs that are the best star candidates and thus can potentially become hubs around which clusters of VMs would be formed. Therefore, it is possible to have ‘VM hives’ spread throughout the cloud in order to respond to specific customer needs.

Note this algorithm can be used as a heuristic approach to solve the clustering problem introduced in Section III-B, however using a technique with much less computational complexity. As can be seen, Algorithm 2 has a complexity that is at least the same that of the Star Algorithm, namely $O(n^2)$, plus additional steps to sort the $centers$ array, which for a Quicksort routine would be $O(n \log n)$ on average.

E. Neighbors Algorithm

Consider now the case where we want to find m pairs of VMs within the cloud which are very close to one another, typically with a round-trip time between them below a given threshold. This is a special case of the Hive Algorithm, but limiting the number of VMs in the hive to only two, in order to avoid an explosion in the number of iterations.

This approach is described in Algorithm 3, which receives a Latency Matrix as input and returns all VM pairs whose RTT is less than the given threshold. As the round-trip time information is symmetric, we only need to test the elements to the right side of the main diagonal ($i < j$, as in Line 2), since the ones to its left will be verified as the loop progresses down the matrix. This still provides a complexity of $O(n^2)$ for the algorithm, however it leaves us with half of the matrix elements to be tested.

Algorithm 3 Neighbors

```
Input: $RTT$ : Latency Matrix for $n$ VMs; $thres$ : max RTT between VMs
1: for $i = 0$ to # rows in $RTT()$ do
2:     for $j = i + 1$ to # columns in $RTT()$ do {only $i < j$}
3:         if $RTT(i, j) < thres$ then
4:             $pairs\{\} \leftarrow RTT(i, j)$
5:         end if
6:     end for
7: end for
8: return $pairs\{\}$
```

IV. Evaluation of End-to-End Policies

A. Reference Architecture

In order to assess the effectiveness of the VM selection policies and algorithms outlined here, we performed extensive simulation studies using a model for inter-VM latencies, which we developed using real measurements of Amazon Web Services (AWS) EC2 instances.

The first step was to define a reference cloud architecture on which to validate the proposed algorithms. In current datacenters, the internal network architecture is usually organized in a hierarchical manner, as shown in Figure 3. On the bottom layer, there are the racks, each with several compute nodes (herein referred to as CPUs, for simplicity) which serve as hosts for a number of virtual machines. VMs on the same CPU communicate directly without any interference from the network. Machines that belong to the same rack but reside on different CPUs need to communicate through a top-of-rack (ToR) switch. VMs on different racks have to use an aggregator to reach each other, for example, if a VM on Rack 1 wishes to send a message to another VM on Rack 2, this message will need to go through the ToR switch of the first rack, then to the second-level aggregator switch, the ToR switch of the destination rack and so on. If the racks belong to different aggregates, then an aggregator on a higher level will also be involved in the communication. Thus, the farther apart are the VMs physically, the higher the communication latency between them tends to be. Although other factors such as host CPU overhead may influence network delay, cloud applications in general can certainly benefit from VM selection policies aiming to assure that communication occurs among VMs close to each other.

For the purposes of validating the proposed locality-aware VM selection algorithms, we employed the cloud architecture described above. In the experiments, it was configured with a top-level aggregator to which 2 other aggregators are connected, each with 2 racks attached to it, with 4 CPUs and 10 virtual machines in each one. This gives a total of 160 VMs in our cloud (10 VMs per CPU, 4 CPUs per rack, 4 racks). Although simple, this reference architecture is consistent with other example cloud architectures reported in the literature and will serve as our testbed.

B. Round-trip Time Sampling

As discussed in Section III-A, the elements in the *Latency Matrix* inform the distance, i.e., the round-trip times between any two VMs within the cloud. For implementation purposes, it is irrelevant whether the data on the matrix refers to the whole cloud or to just a subset of VMs allocated from the cloud provider to a customer. In the context of the Recommender Service (Section II-B), it is more realistic to assume we are dealing with a subset of VMs in control of a cloud services broker (Figure 1) that will utilize the proposed algorithms to make an informed selection of VMs and relay them to an interested cloud customer.

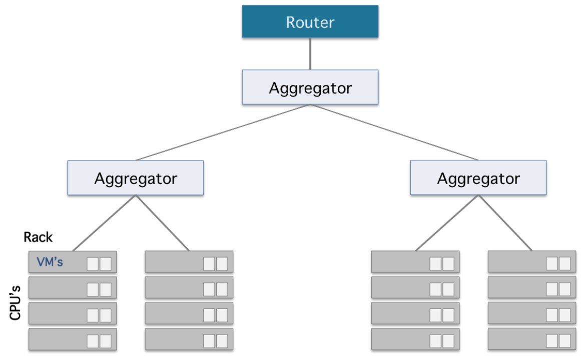

Fig. 3. Cloud architecture used as reference for algorithm validation

TABLE I. Different levels for round-trip times

Level	VM Location	Weight	%
1	Same CPU	0.1	5.6%
2	Same Rack	1	18.8%
3	Same Aggregate	10	25.1%
4	Distinct Aggregates	50	50.3%

To populate the Latency Matrix, we employed a uniform distribution in the interval $[0, 1)$ to randomly generate the round-trip time values. Those served subsequently as input for a cumulative distribution function (CDF) to produce more realistic RTT figures, consistent with the different RTT thresholds found in a typical cloud. As observed, RTT values do not vary uniformly over an interval, but rather form small groups of close values depending on whether the communicating machines are found on the same CPU, rack, aggregate, and so on.

Table I lists the four different RTT levels considered depending on where the VMs are located. The weights attributed to each level were consistent with the experiments performed on Amazon EC2 machines, as reported in Section III-A. The last column tells the percentage of VMs found in each category, derived from the architecture shown in Figure 3. This table is eventually used to populate the Latency Matrix.

C. Results

In this section, we will report some results obtained by employing our policies to make a selection of m VMs out a of a population of n machines. The main input to the selection algorithms is the Latency Matrix as described in sections III-A and IV-B. Each algorithm is compared with the random case, i.e., when a customer simply assigns applications to VMs at random, as usually happens in cloud settings nowadays.

In the first experiment, we analyze the performance of the **Neighbors Algorithm** against a random selection of VMs. The

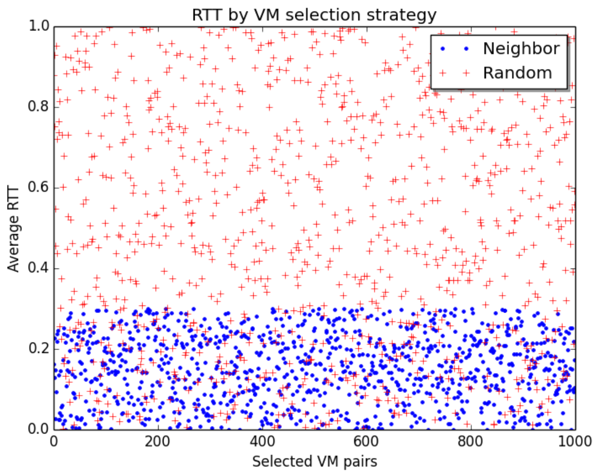

Fig. 4. Neighbors vs. Random selection

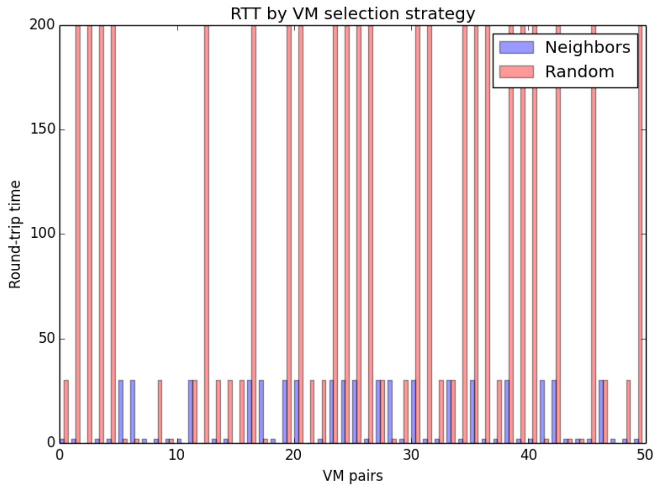

Fig. 5. Neighbors vs. Random selection (same aggregate)

Latency Matrix is populated using a uniform distribution in the interval $[0, 1)$ and the RTT threshold is set at 0.3 units of time. We ran the algorithm for 100 VMs (a 100×100 RTT matrix) and searched for 1,000 VM pairs within the given threshold. Our results in Figure 4 show that the RTT values for all VM pairs selected through the Neighbors Algorithm fall stricly in the 0.0 to 0.3 range, as expected, with an average time of 0.1553. For the random case, the selected pairs are scattered throughout the solution space, since any one of them could be selected, and an average RTT of 0.5073 is reached. The percentage of discarded pairs by the Neighbors approach in order to achieve the desired selection is 70.88% on average, which is consistent with the fact that we are using a uniform distribution between 0 and 1 and the cutting point is set at 0.3.

For a more realistic view of our algorithm in action, we now generate a Latency Matrix according to the CDF detailed in Table I. The RTT threshold is set at 40 so that only VMs pairs located in the same aggregate, rack or CPU are selected by the algorithm. The objective is again to find 1,000 pairs of machines whose RTTs satisfy the specified criteria. In this experiment, as can be seen through the extract in Figure 5, RTTs are grouped around the thresholds specified by the CDF, namely $\{0.1, 2, 30, 200\}$ $(level \times weight)$. The average RTT for VM pairs selected by the Neighbors algorithm is 16.9566, compared to an average of 112.6947 when a random selection is used. The percentage of discards by the Neighbors approach is 49.44% on average, corresponding to roughly 50% of VMs located in distinct aggregates, as defined in Table I. For all cases reported the algorithm execution time was less than 1 second.

The second selection policy to be analyzed is the **Star Algorithm**. In this case, the RTT values for the Latency Matrix were generated according to the CDF described previously. We consider 100 VMs and run the algorithm to find the VM that has the lowest RTT average concerning all the others. In Figure 6, there is a comparison between the real center found by the Star Algorithm and 1,000 random choices of centers. As can be seen, our algorithm always points out the same center for a given RTT Matrix setting, here with an RTT average of 87.4828. However, in the random case, most "centers" fall far

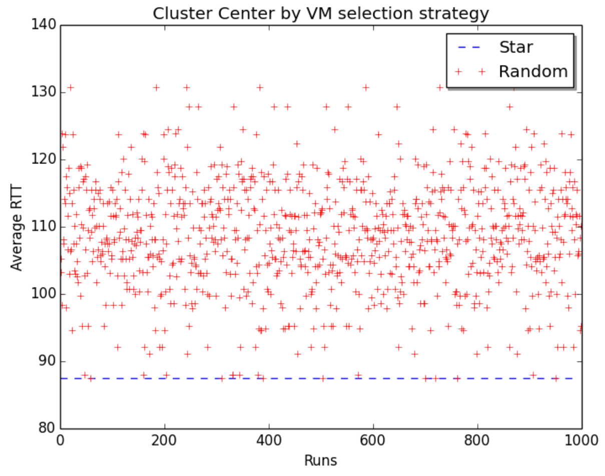

Fig. 6. Star vs. Random center selection

from the target, which they miss by more than 20%.

Finally, we performed an experiment to study the **Best Centers** Algorithm. We generated a Latency Matrix with 100 VMs using a CDF as done previously. This algorithm caters for a special class of applications that require a set of hubs that somehow condense the traffic around them, functioning as local points of attraction. In this experiment, we aim to find the 10% best centers out of the set of VMs provided. Each VM is individually chosen as a center candidate and its average RTT is computed. Finally, all average RTTs are sorted from lowest to highest. The results are depicted in Figure 7 where the blue triangles precisely point out which VMs are better suited for the purpose of being local hubs. Contrary to what may seem at first, those best centers are not clustered in any special region of the matrix but rather scattered on it; for instance, in one particular run of the algorithm the list of center candidates returned was $\{36, 57, 25, 5, 82, 77, 50, 45, 75, 56\}$. The execution time of the algorithm was less than one second, what makes it fast enough to be used in a real cloud setting.

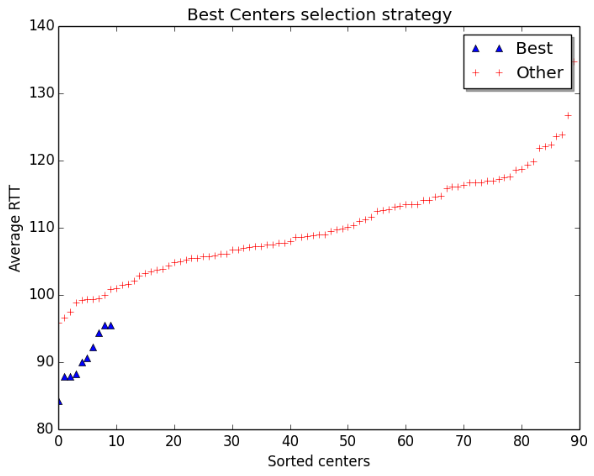

Fig. 7. Best Centers selection

D. Discussion

As the above results have shown, the employment of more informed VM selection policies allows the cloud customer to better match the application's profile to current cloud and VM utilization. This results in a better application-to-VM mapping and can have a decisive impact on the performance of applications executing in the cloud. All our proposed algorithms have reached their objective, with negligible execution times. Additionally, they were able to choose VM assortments far better than the ones obtained by making a blind (or random) resource selection, as is normally the case.

Note that a simple and straightforward selection policy as the Neighbors algorithm can indeed yield very good results in terms of extracting a subset of VMs that complies to some previous requirements specified by the cloud customer. This algorithm can even be fine tuned so as to return VMs exhibiting a given latency from each other, as it was the case with the 'same aggregate' experiment above. However, although delays are strongly correlated with physical distance within the cloud, it may not always be the case because sometimes higher latencies are due to CPU overhead while processing the networking part of the VM. In anyway, what is needed from the the customer's viewpoint are VM pairs with a delay within a given time span and this is precisely what the algorithm provides. Moreover, its small execution time allows for frequent runs in order to quickly respond to ever changing application communication patterns within the cloud.

Likewise, the Star Algorithm results are an excellent example of how an informed VM selection can make a huge difference and can really empower cloud customers. Suppose that, for some reason, they needed to select one of their contracted VMs to be the center of a particular subset, perhaps because that machine should run a directory service. Most probably, our customer would end up, by misfortune, allocating the directory server far from its clients, with a crucial impact on the system's performance. On the other hand, our algorithm can precisely indicate the proper VM for such a role. Better yet, it achieves this conclusion by using RTT measurements only, with no further knowledge of the cloud's internal architecture or of any other third-party applications concurrently running on those VMs.

The Best Centers algorithm accomplishes a far more difficult task, that of finding a set of VMs, among the ones contracted, better suited to perform the role of local traffic hubs. Such a feature would be specially important in a peer-to-peer application within the cloud, where each peer node could assume the role of a supernode for its neighboring machines. As our results demonstrated, this algorithm is able to sort out the exact VMs to this end and turn them in to the cloud customer, a tremendous improvement over a blind VM selection.

The selection policies and accompanying algorithms analyzed herein distinguish themselves from similar approaches because they are able to make better VM commitments with minimum knowledge of the underlying cloud infrastructure. They simply infer VM relative distances from round-trip time measurements and employ this information as heuristics to make wiser VM selections.

V. Related Work

This paper considers the problem of VM selection and allocation in cloud infrastructures. This subject has been the focus of much research for several years and the issue addressed here is challenging: given a certain workload and/or application to be executed in a cloud, we need to decide which machines to choose for this task. In particular, in this paper we deal with locality-aware VM placement, as several studies have shown that network delays may play an important part in overall application completion times, affecting cloud customers and providers alike.

The paper by Alicherry [3] advocates that the best application performance is obtained when VMs are located closer to each other in the cloud, preferably in the same rack, therefore they try to minimize datacenter fragmentation in order to avoid inter-rack traffic. They approach the issue of VM allocation as a graph partitioning problem and indicate it is usually NP-hard, hence they use some approximations to find the solution to the problem [12], [13]. However, their work assumes some knowledge about VM locations inside the datacenter, what may prevent the application of this approach in public clouds such as Amazon EC2, which are totally opaque to their users.

In [4], the authors also take network usage into consideration when attempting to allocate VMs, but they are particularly interested in data intensive applications. Their paper employs a data access matrix that tells the access time from each physical machine to the related data. The latency-aware placement and migration algorithms proposed by the authors are modeled as an optimization problem and are validated on the CloudSim platform.

Meng el al. [2] study the problem of traffic-aware virtual machine placement in different cloud topologies [14], [15] and also model it as an optimization problem. Here the objective is to minimize communication costs by placing VMs with larger communication demands closer to each other. They utilize traffic traces from huge datacenters in their analysis and further perform a comparison on the impact of the traffic patterns and the network architectures on the potential performance gain of traffic-aware VM placement.

LaCurts et al. [5] propose a comprehensive approach for network-aware VM allocation by resorting to network measurements and application profiling in order to find a better solution for application-to-VM mapping. They make extensive measurements on Amazon and HP clouds and formulate an elaborate overall solution. However, as the authors admit, their approach is tailored to current hierarchical cloud architectures and thus they cannot guarantee its efficiency if those topologies change in the future.

In Oktopus, Ballani et al. [16] create a virtual topology on top of the real cloud and places VMs according to bandwidth and oversubscription demands. Here the main objective is to predict the completion time of tasks rather than finding an optimal placement of VMs. However, cloud providers have to agree to deploy this system on their infrastructures, what diminishes this solution's usefulness in practical terms.

In [17], the authors discuss an optimized allocation approach for MapReduce jobs, therefore they produce an application-specific rather than general approach. Moreover, their solution relies on some knowledge of the network topology, what prevents its application in public clouds.

The main difference between our approach and these works is that our solution uses an end-to-end approach that is completely independent from datacenter topology, is based on lightweight and viable round-trip time measurements, does not rely on any proprietary protocol to be deployed in the cloud, is not dependent on cloud provider acquiescence, and is flexible and extensible to different types of applications and workloads.

VI. Conclusion

The problem of VM selection and placement in current IaaS clouds constitutes a challenging research topic, specially considering the enormous proportions of clouds nowadays and the advent of novel latency-sensitive applications that place new demands on current infrastructures. Moreover, recent research has pointed out that intercommunication patterns play a major role in application overall performance in the cloud, thus this topic has drawn much attention from the community.

In this paper, we proposed a Recommender Service to which a cloud customer can make a request for VMs that should meet some particular criteria and receive, in return, a set of machines that satisfies those demands. This service allows for knowledgeable VM allocation decisions, in contrast to the blind cloud resource allocation seen nowadays.

In particular, we focused our study on locality-aware VM selection policies and algorithms that utilize latency between VMs as a heuristic to support allocation decisions. Round-trip time measurements were used as the main input to the algorithms and four different VM selection policies were proposed. All of them employ an end-to-end approach and are independent from datacenter topology details.

The first one, the Hive algorithm, was found to be NP-hard and thus we produced a simplified version of it, named the Neighbors algorithm. This latter approach returns pairs of nearby VMs in the cloud whose latency is strictly below a given threshold. It can also be fine-tuned in order to return only VM pairs in the same rack, for example, using latency as a metric to infer probable location in the cloud. This algorithm exhibited a good performance even when searching for 1,000 VM pairs in a 100×100 matrix.

The Star algorithm, another policy evaluated in this paper, is intended to find the VM located in the midpoint of a subset of VMs, in terms of average RTT distance. In one experiment, this algorithm was able to point out the center among 100 VMs and that was compared to 1,000 other random choices. The Star algorithm outperformed them in the vast majority of cases, always identifying the right midpoint.

Finally, through the Best Centers algorithm, we approached the more general problem of a customer's application that requires several local centers throughout the cloud. In one experiment, this algorithm was used to select the 10% best VM centers out of a Latency Matrix of 100 VMs (or 10,000 elements), what was accomplished in just a second. And, as intended, the resulting subset of VM centers was well distributed in the cloud.

As future work, we intend to deploy the Recommender Service and related algorithms on a real cloud infrastructure and also to evaluate the usefulness of this approach for latency-sensitive applications. This will allow for a better understanding of our algorithms' performance in a real-world setting as well as indicate novel forms of applying them.

Acknowledgment

The first author would like to thank CNPq, Brazilian National Council for Scientific Development, for its financial support and also the Hariri Institute at Boston University for having received him as a Visiting Fellow for the year of 2014. This work was supported in part by a number of NSF awards, including PFI:BIC award #1430145, SaTC awards #1414119 and #1012798, CNS awards #1347522 and #1239021.

References

[1] A. Bestavros and O. Krieger, "Toward an open cloud marketplace: Vision and first steps," *Internet Computing, IEEE*, vol. 18, no. 1, pp. 72–77, 2014.

[2] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[3] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 963–971.

[4] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *9th International Conference on Grid and Cooperative Computing (GCC), 2010*. IEEE, 2010, pp. 87–92.

[5] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan, "Choreo: network-aware task placement for cloud applications," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 191–204.

[6] MOC, "Massachusetts Open Cloud (MOC) ," http://www.bu.edu/moc, 2014, [Online; accessed 3-Nov-2014].

[7] M. Böhm, G. Koleva, S. Leimeister, C. Riedl, and H. Krcmar, "Towards a generic value network for cloud computing," in *Economics of Grids, Clouds, Systems, and Services (GECON)*. Springer, 2010, pp. 129–140.

[8] D. Battré, N. Frejnik, S. Goel, O. Kao, and D. Warneke, "Evaluation of network topology inference in opaque compute clouds through end-to-end measurements," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 17–24.

[9] R. Shea, F. Wang, H. Wang, and J. Liu, “A deep investigation into network performance in virtual machine based cloud environments,” in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1285–93.

[10] B. Jonglez, M. Boutier, and J. Chroboczek, “A delay-based routing metric,” *arXiv:1403.3488*, 2014.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 3rd ed. MIT press Cambridge, 2009.

[12] N. Guttmann-Beck and R. Hassin, “Approximation algorithms for minimum k-cut,” *Algorithmica*, vol. 27, no. 2, pp. 198–207, 2000.

[13] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[14] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[15] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[16] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, “Towards predictable datacenter networks,” in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 242–253.

[17] B. Palanisamy, A. Singh, L. Liu, and B. Jain, “Purlieus: locality-aware resource allocation for mapreduce in a cloud,” in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 58.