CloudAffinity: A Framework For Matching Servers to Cloudmates

Marcos D. Assunção, Marco A. S. Netto, Brian Peterson
Lakshminarayanan Renganarayana, John Rofrano, Chris Ward, Chris Young

IBM Research

{marcosda, mstelmar}@br.ibm.com, {blpeters,lrengan,rofrano,cw1,ccyoung}@us.ibm.com

***Abstract*—Increasingly organizations are considering moving their workloads to clouds to take advantage of the anticipated benefits of a more cost effective and agile IT infrastructure. A key component of a cloud service, as it is exposed to the consumer, is the published selection of instance resource configurations (CPU, memory, and disk). The number of instance configurations, as well as the specific values that characterize them, form important decisions for the cloud service provider. This paper explores these resource configurations; examines how well a traditional data center fits into the cloud model from a resource allocation perspective; and proposes a framework, named CloudAffinity, aimed at selecting an optimal number of configurations based on customer requirements.**

I. Introduction

In an effort to reduce IT capital and operational expenditures, organizations of all sizes are moving their workloads to the cloud. Clouds come in many shapes and sizes, but share a common foundational technology, virtualization. Virtualization provides the ability to host multiple distinct operating systems on a single hardware unit. This along with a well defined interface (graphical or programmatic) allows cloud customers to freely allocate, deallocate and consume the underlying computing capacity.

This change in IT strategy risks replicating traditional physical IT challenges (*i.e.* underutilized servers and physical sprawl) to virtual challenges (overprovisioning of virtual resources and image sprawl [1]). Providers of cloud services must be cognizant and careful not to repeat the same mistakes in the virtualized world. Strong governance and well defined interfaces are a needed to ensure that the "mess" is kept outside the cloud.

In order to reduce this risk, cloud providers have currently taken an approach which simplifies the IT experience. One example of this approach is the practice of the cloud provider publishing a limited set of virtual hardware combinations from which the consumer must select. This approach is a far cry from the current state of procuring a physical server and the multitude of configurations available for selection. This paper refers to the combination of virtual CPU, memory, and disk as a *virtual resource template* (VRT) or just *template*. The concept is distinct from the software stack which is run within the resource container and is commonly referred to as a *virtual machine template*, *golden image*, *clone source* or *master image*.

The ability for a customer to rapidly deploy new workloads via new cloud instances is a key selling point of cloud and virtualization. However, for this to be a reality customers must find an appropriate set of templates that meet the minimum requirements of their application workload; including non-functional requirements such as speed, robustness, and scalability. This selection process is largely manual and expects customers to tediously iterate through many choices to find the one most suitable for their needs.

This paper examines hundreds of servers within a real enterprise data center and investigates the cost implications of migrating their associated workloads into a cloud. The server configurations are considered as an aggregate of a specific workload's minimum requirements. The workload represents an entire stack including the operating system, middleware, application, and the supporting non-functional requirements (*e.g.* ability to perform x transactions per second). Using the same data set, we propose a method to evaluate the optimal number of templates required to minimize the excess resource allocations associated with the standard cloud approaches of small, medium and large. More specifically, the contributions of the paper are:

- Exploration of a challenge faced when moving a large workload into a cloud with a fixed set of virtual resource templates;
- Evaluation of the matching of an enterprise workload into a well known public cloud (Amazon's EC2), which considers costs due to overprovisioning. The evaluation raises awareness that there are implications of the state of the art (small, medium, and large) and many improvements must be realized before cloud can be a utility;
- A framework, called CloudAffinity, aimed at finding an optimal set of virtual resource templates that can meet the requirements of an existing data center.

The solutions, results, and discussions presented in this paper can be leveraged by researchers and practitioners working on the cloud computing space, in particular on management of VM templates.

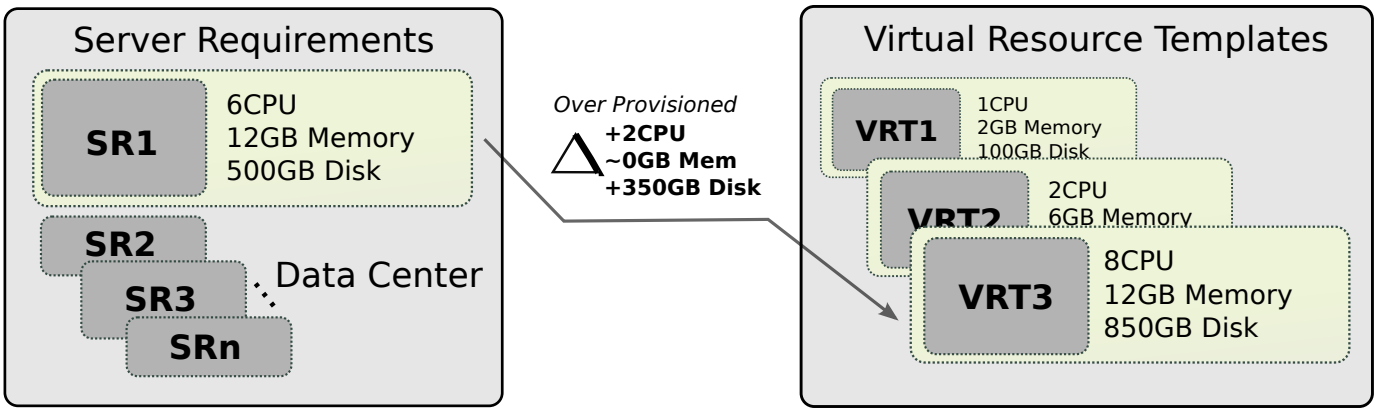


Fig. 1. Traditional template lifecycle. On the left-hand-side multiple server requirements define the existing servers within a data center. An optimal image—defined by at least the existing requirements with the least amount of waste—is selected and the delta is calculated to represent the amount of over provisioned resources.

II. Problem Description and Metrics

A. Problem description

Organizations are familiar with the process of sizing and procuring server hardware to meet the needs of a specific workload. This process involves tasks such as selecting vendors, hardware architecture, amount and type of CPU, memory, and disks. However, this build-to-order model has changed with the advent of the cloud. Cloud abstracts the underlying complexity of the infrastructure and only exposes a simplified view to the consumer. The current manifestation of this is seen with the concept of small, medium, and large cloud templates (Figure 1). This abstraction allows the cloud to provide a level of simplicity akin to ordering a fountain pop at a fast-food chain. However, this approach also implies that workloads scale uniformly across CPU, memory, and disk bounds.

Table I summarizes virtual resource templates offered by cloud providers, highlighting the significant variation between the number of templates offered and the specific parameters for each template. For instance, Amazon EC2 offers additional templates geared towards workloads with disproportionate CPU and memory requirements. Many factors motivate these variations and are ultimately determined by the cloud consumer and cloud provider. For example, cloud consumers are concerned with allocating the fewest resources necessary to keep costs to a minimum and still meet the requirements of the workload. Consumers are also interested in having an easy and effective way to search for the best template based on their unique requirements (this problem is similar to finding an appropriate image [3]). From the cloud provider perspective, the goal is to offer the most appropriate options for the consumer without creating unnecessary overhead and complexity (hundreds or thousands of CPU, memory, and disk combinations would force an evaluation of the word "template").

These challenges highlight the need for cloud providers to have a method to derive the optimal set of templates for a given workload. In a public cloud environment this workload may be ill-defined and hard to measure due to the diversity of consumers and uses. However, in the scenario that an organization is internally looking to move workload into a private cloud, there is necessary access to the existing server configurations which can be used to influence the selection of templates. In both models defining templates is crucial to the success of cloud and the ability to migrate existing workloads into it with minimal overprovisioning.

B. Metrics

The effectiveness of the matching algorithms evaluated in the creation of this paper was measured against the following set of metrics:

- **Cost:** amount (in dollars) paid to maintain a cloud instance based on a given resource template;
- **Euclidean distance:** distance between two template definitions. In our case we consider CPU, memory, and disk to calculate the distance between the provider's template and the user's requirements;
- **Matching factor:** Difference (percentage) between the user's requirements and what the template has to offer, for each virtual resource (CPU, memory, disk).

III. Motivation Scenario

The to-be migrated workload analyzed here is based on a set of server configurations from a real data center. This data center requires that all workloads submit a detailed document of server requirements (software and hardware). These requirements are ultimately used by the various delivery teams to procure and provision the necessary hardware and software configurations. This freedom generates a set of configurations that cannot be re-utilized by other users due to their fine grained specifications. Clearly this scenario results in minimal overprovisioning of resources as users are provided with exactly what they require. This contrasts with the cloud

TABLE I
CLOUD PROVIDER VIRTUAL RESOURCE TEMPLATES.

Cloud Provider	Template*				
	micro	**small**	**medium**	**large**	**extra-large**
Amazon EC2[1]	?/613/0**	1/2/160		4/7/850	8/15/1690
Softlayer CloudLayer[2]	1/1/25	1/1/100	2/2/100	4/4/100	8/8/100
OpSource Cloud[3]		1/2/80	2/4/160	8/16/640	
IBM Research Compute Cloud [2]		1/1/18	2/2/36	4/6/72	4/12/72

*Templates are defined by: number of CPUs/memory (GB)/disk (GB).
**The micro instance within Amazon does not define a specific amount of CPU.
[1]http://aws.amazon.com/ec2/instance-types/
[2]http://www.softlayer.com/cloudlayer/computing/
[3]http://www.opsource.net/Services/Cloud-Hosting/Pricing

approach where each of these workloads would certainly not fit perfectly within a small set of templates. In this context, we envision providers asking customers for exactly what they want and then provisioning from a known selection. As time progresses and the number of requests increases, providers will be able to make better instance types available for customer selection. As well, this could be seen from the perspective of a third party value-add provider who would broker the requirements of customers and clouds to help effectively match workloads and influence the destination configurations.

In order to evaluate this trade-off between flexibility and resource waste, as a motivating scenario, we first analyze how well the workload fits into the current set of Amazon EC2 templates (referred to as *Instance Types* by Amazon).

A. Workload overview

The data set used in this paper involves a selection of 747 enterprise server configurations, which include specifications for CPU, memory, and disk. All server requirements are based on the same operating system and hardware platform to rule out differences not attributable to the workload. The documented server requirements span a three year time period and have all been collected with the same tool, providing consistent and comparable specifications. Figure 2 shows the general breakdown of the existing servers by workload classification.

This data set accurately represents the underlying hardware resources which are required to support a large number of enterprise workloads. Migrating these workloads to a cloud involves instantiating templates that cover the needs of all of the existing servers. This is an accurate scenario for an organization who is looking to move their workload into a private cloud (same hardware architecture, same operating system, and same organization). However, due to the variations in used configurations and the size of the data set, it could also be considered as a general representation of cloud candidate workloads.

B. Workload migration into Amazon's EC2 templates

Two Euclidean distance based approaches were used to compare the enterprise server requirements and Amazon EC2 templates. The first approach is *strict*, and only looks for Amazon EC2 instances that can meet all the server requirements. The second approach, called *flexible*, considers instances that do not meet all the server requirements. We also looked at the distance considering two (CPU and memory) and three (CPU, memory, and disk) server requirements. The disk values were excluded from the two server requirement sets due to their relatively large magnitudes and the impact that created for the comparison.

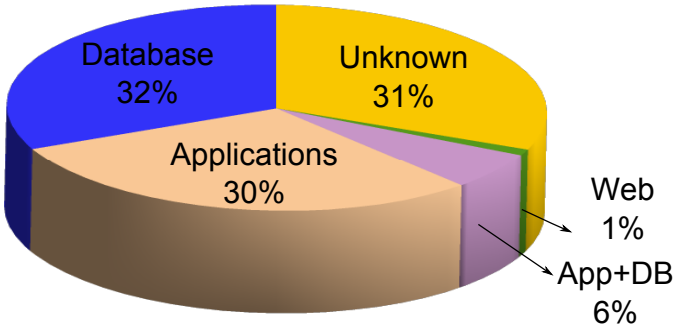
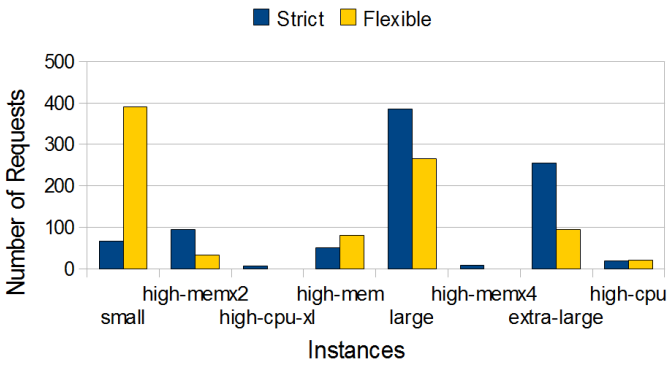
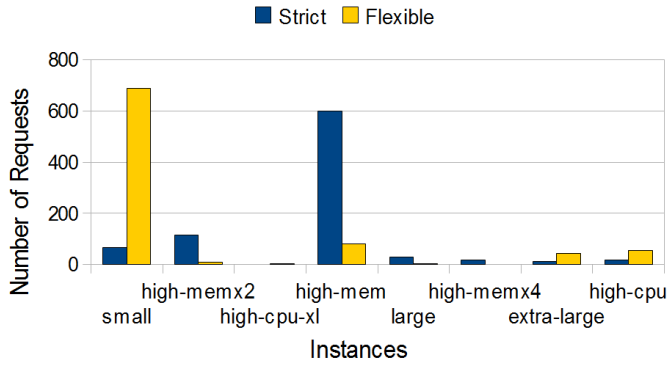


Fig. 2. Workload classifications for each server. "Unknown" represents server requirements that do not include any software specifications.

Figures 3 (a) and (b) present the number of server requirements that can be mapped to an Amazon EC2 template using two and three comparison parameters respectively. Both figures show that there is a considerable difference between the strict and flexible approaches. The difference is due to the fact that the instance templates cannot have a direct match against the server requirements. By considering disk as a comparison parameter, it is noticeable that the main difference occurs with instances that have higher computing power capabilities. The results are more visible for the matching factor metric (Figure 4); a metric that shows the difference between the server requirements and Amazon templates for each hardware requirement where disk plays an important factor for the comparison.



(a) CPU and memory as comparison parameters.



(b) CPU, memory, and disk as comparison parameters.

Fig. 3. Requests per Amazon EC2 template.

The rest of this paper focuses on the first approach (*strict*) which considers a known set of minimum workload requirements that are to be migrated into a cloud.

IV. Workload Requirement Analysis

One method to select the optimal number of templates relies on the analysis of known server configurations. This option requires an accessible set of representative configurations which could be obtained from a couple of sources.

As previously discussed, the first source involves a documented set of workload requirements which typically are created by an application architect. For example, if an architect were to deploy a LAMP (Linux, Apache, MySQL and PHP) workload to satisfy 2000 page views an hour they must first allocate minimum resource requirements for each software component and then overlay additional resources to meet the usage of the application. These aggregated requirements can be viewed as a representation of the exact server configuration required to meet the workload's needs. In the traditional IT delivery scenario this request is passed to the IT provider who then proceeds to procure and configure the necessary hardware components.

The second source involves examining an existing set of deployed servers to discover their resource allocations. This approach is advantageous as it allows actual representations to be used versus idealistic definitions from the architect.

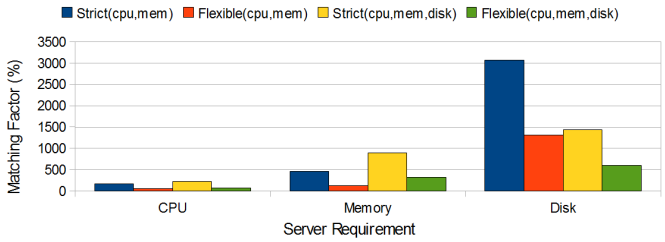


Fig. 4. Matching factor for server requirements (*i.e.* CPU, memory, and disk) against Amazon EC2 instances.

The downside is that it is more difficult to partition specific workload requirements from the natural "accumulation" of processes running on the server. This data may tend to overestimate the requirements for applications due to the practice of allocating but rarely de-allocating resources.

The following section applies a clustering approach to data gathered using the first source in an effort to find natural resource groupings.

A. Geometric clustering

Given a set of server requirements, a natural way to identify templates is to view the server requirements as points in n-dimensions and apply standard clustering techniques such as K-means clustering. However, before we can apply any such clustering techniques we need to first define a similarity function between the requirements to quantify how similar (or close) two given requirements are. If we view the three attributes of server requirements, CPU, memory, and disk, as coordinates in a 3D space, then similarity can be defined as the Euclidean distance between any two points in this 3D space. But, such an Euclidean distance would give equal weights to all the attributes and discard the following two important properties of the data set: (i) the typical values for disk attributes are 10 times higher than that of the memory and (ii) the dollar cost for 1 GB of disk is 10 to 15 times smaller than 1GB of memory.

Values for each attribute have been normalized as follows: from each value we subtract the mean and divide it by the mean absolute deviation. This approach handles issue (i) above and provides a set of values scaled between $[0, 1]$. This normalization does not handle issue (ii) which is addressed in the following section. We performed such a normalization and then applied K-means clustering for a range (2 to 25) of clusters. In each clustering trial, we tried 10 randomly selected starting cluster-centers. We did not find any meaningful clusters which was counter to our initial intuition. Digging a bit deeper we found that the data does not have any natural clusters. To illustrate this nature of the server requirements we show in Figure 5 the memory and CPU attributes of the server requirements. It is easy to observe that there is a weak correlation and there are no strong clusters present in the data. This lack of any natural clusters is also reflected when we consider all the three attributes (CPU, memory, and

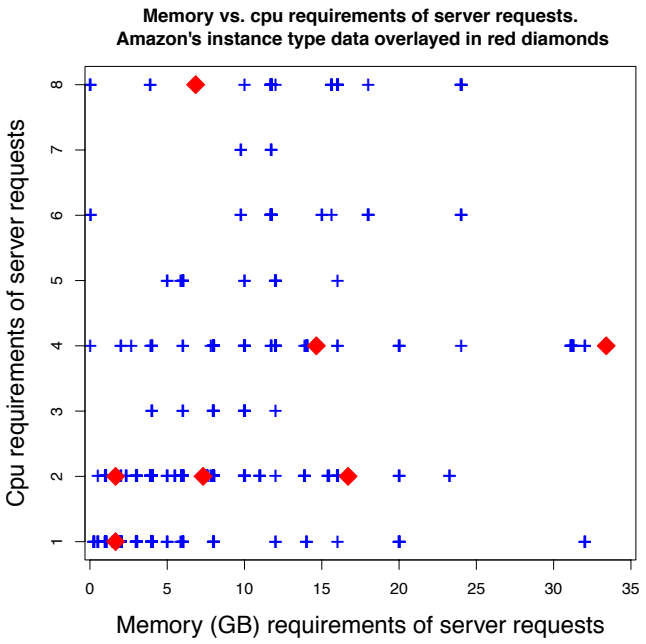


Fig. 5. Memory vs. CPU scatter plot.

disk) together. As reference points we included in Figure 5 the CPU and memory attributes of the Amazon templates in red diamonds.[1] This finding also supports an earlier question with respect to the validity of scaling resources of small, medium, and large templates in unison. If such a uniform scaling of the three attributes is valid then the scatter plot would have a clear trend line and show a positive correlation between values.

To summarize, K-means style clustering does not help in identifying templates due to two important reasons. First, there are no natural clusters in the data—the server requirement values for the CPU, memory, and disk attributes are widely spread out. Second, using Euclidean distance to quantify similarity between server requirements completely ignores the relative dollar cost differences between the attributes. In the next section, we propose a scheme that overcomes these two shortcomings.

Overall this was a surprising result. We initially expected to see clear clusters in the data which would support the notion of few, well defined templates. We found the fact that there was poor correlation between all three resources especially interesting and counter intuitive.

V. Template Selection Considering Cost

This section, which embodies the core of the *CloudAffinity* framework, presents a cost aware clustering scheme to directly select a small number of templates to satisfy the server requirements. It also provides techniques that can be used by a cloud provider to analyze the trade-offs between the number of templates, number of server requirements satisfied, and cost of satisfying these requirements.

The key idea behind the cost aware clustering is to first calculate a minimum cost match for each server requirement. This process provides a set of most frequently used templates which are used to determine the final, smaller, set of templates and the total cost. The CloudAffinity framework currently considers CPU, memory, and disk size as server requirements. These parameters are limited to the current state of cloud and represent an approach that is applicable to today's industry state of the art. Although the additional parameters (such as network) were not considered, CloudAffinity can handle additional parameters to generate template sets.

Figures 6 and 7 present the outline of the cost aware clustering scheme and its auxiliary functions respectively. The selection of best templates is performed as follows:

1) We compute the minimum match for each server requirement by matching CPU, memory, and disk to the smallest values in the corresponding target lists. The function `findMinMatch` in Figure 7 describes how the minimum match is computed. Using the minimum match for each server requirement also allows us to compute the minimum cost. Such cost is a lower bound on the cost for satisfying a server requirement request based on the target CPU, memory, and disk values provided, and is used later on to determine whether a request can be satisfied at its minimum cost. We refer to these matches as *minimum cost match.*
2) Given the minimum cost matches from the previous step, we now proceed to derive a set of templates. A template is a combination of the CPU, memory, and disk values from the targets considered in the previous steps. Note that a single template can satisfy more than one server requirement request and some templates may not be used to satisfy any request. The set of templates that satisfy all the requests is computed by selecting the set of unique templates from the matches computed in the previous step. The function `deriveTemplates` described in Figure 7 outlines the steps involved in deriving the templates. For our set of requests and the target resources, we found that a set of 89 templates is sufficient to satisfy all the requests with minimum cost.
3) A natural way to summarize the derived templates is to sort them by the number of requests satisfied by each one. This is achieved in the next step and Figure 8 shows the results graphically. Each bar represents a template and the value of the bar (shown on the top) represents the number of satisfied requests. The label of each bar shows the CPU, memory and disk values of that template. It is interesting to note that the top 10 templates satisfy 55% of the requests at minimum cost and the top 20 templates satisfy 71% at minimum cost. This seems quite reasonable to have such large percentages of server requirements met with minimum cost by a relatively small number of templates. This might lead us to believe

[1]To illustrate the structure of the data, we have omitted one Amazon template (high-mem-x4) whose large values (CPU=8, memory=26, disk=66.80) skew the plot and compress all the values to the left.

```
// Values for cpu, memory, and disk used for the results
// reported in this paper. The proposed scheme can be
// directly used with any values for these attributes.
// targetCpuList = (1,2,4,8,12,16,32);
// targetMemList = (1,2,4,8,12,16,32,64,128);
// targetDiskList = (100,200,300,400,500,600,700,800,900,
//                   1000, 1200,1400,1600,1800,2000);
// topKList = (5,10,15,20,25)
//
// Cost for one cpu, one GB of memory or disk
// CpuCost =  119.86  MemCost = 180.2  DiskCost = 1.34

// Selects the top k templates for a set of server
// requirement requests, target cpu, memory, and disk
// values, and values for (top) k.
SelectBestTemplates (requests, targetCpuList,
                     targetMemList, targetDiskList,
                     topKList) {

  // for each request find the minimum cost match
  [matches, globalMinCostList] =
    findMinMatch(requests, targetCpuList,
                 targetMemList, targetDiskList)

  // from the matches collect templates with at least
  // one match and discard templates with no any match
  [templates, numRequestsSatisfied] =
    deriveTemplates(matches, globalMinCostList, requests)

  // sort templates by number of satisfied requests
  [sortedTemplates, sortedCount] =
    sort(templates, numRequestsSatsified)

  // for each (top) k value, compute the matches
  for ( i in 1 ... length(topKList) ) {
    topK = topKList[i]
    // match requests to a set of top-K templates
    [ minCostMatches[i], higherCostMatches[i],
      unsatisfied[i] ] =
    calcCostForTopTemplates(sortedTemplates[1:topK],
                            globalMinCostList)
  }
}
```

Fig. 6. Pseudo code for selecting the top k templates. Figure 7 presents the auxiliary functions.

```
// Cost for one cpu, one GB of memory or disk
// used in our analysis
// CpuCost =  119.86  MemCost = 180.2  DiskCost = 1.34
//
// for each request find the minimum resource match
// and compute the minimum cost for it
findMinMatch(requests, targetCpuList, targetMemList,
             targetDiskList) {

 int numRequests = numRows(requests)
 int matches[numRequests][3]
 for ( i = 1 ... numRequests ) {
   // minMatch() finds the minimum value from the targetList
   // that satisfies the request. Note that the satisfying
   // resource value is greater than or equal to the request
   matches[i][1] = minMatch(requests[i][1], targetCpuList)
   matches[i][2] = minMatch(requests[i][2], targetMemList)
   matches[i][3] = minMatch(requests[i][3], targetDiskList)
   cost[i] = matches[i][1] * CpuCost +
             matches[i][2] * MemCost +
             matches[i][3] * DiskCost
 }
 // returns a pair of lists
 return ( [matches, cost] )
}

// Given a list of matches find the set of unique
// templates that match all the requests
deriveTemplates(matches ) {

 for ( i = 1 to numRequests ) {
    // create a string that encodes the matching templates
    matchStrList[i] =
      paste(matches[i][1], matches[i][2], matches[i][3])
 }

 // findUnique returns a set of unique values in
 // matchStrList and for each unique value how many times
 // it occurs in matchStrList
 [templatesList, countList] = findUnique(matchStrList)
 return ( [templatesList, countList] )
}

// Input: templates, requests and their cost
//        for minimum cost match
// For a given set of templates check if a request can be
// satisfied, and if so, compute whether it can be satisifed
// with min cost or higher cost. Also, compute the number of
// unsatisfied requests.
calcCostForTopTemplates(templates, requests,
                        globalMinCostList) {

 int minCostMatches = higherCostMatches = unsatisfied = 0;
 for ( i = 1 ... numRequests ) {
  // given a set of templates, minCostMatch() finds a
  // minimum cost match for a request. Returns -1 for index
  // and a 0 for cost when a match is not found
  [ matchIndex[i], cost[i] ] =
     minCostMatch(requests[i], templates)
  if ( matchIndex[i] == -1 ) {
    unsatisfied = unsatisfied + 1
  } else if ( cost[i] > globalMinCostList[i] ) {
    higherCostMatches = higherCostMatches + 1
  } else {
    minCostMatches = minCostMatches + 1
   }
 }
 return ( [minCostMatches, higherCostMatches, unsatisfied] )
}
```

Fig. 7. Pseudo code for auxiliary functions used to calculate top k templates.

that there is a "cloud appropriate" subset of workload within the overall data set.

4) With the sorted set of templates, we can now do interesting trade-off analyses. To start with, we can as the following questions: "If we consider only the top (most used) 10 templates, how many requests are satisfied? And, how many of them are satisfied with minimum cost? How many with higher cost?" The next step in the scheme computes the answers to above questions for a set of most used templates. It takes as input the k values $\{5, 10, 15, 20, 25\}$ (from `topKList`) and computes the matches for each top k templates. The function `calcCostForTopTemplates` described in Figure 7 shows how the matching is done. The results are presented in Figure 9.

The results summarized in Figures 8 and 9 provide a basis for a private cloud provider to quantitatively analyze a set of server requirements, understand the most used templates, and choose an optimal set of templates.

A. Cost vs. number of templates: trade-off analysis

Table II shows a summary of the total cost of satisfied server requirements for a given set of top k templates. Also, the last column presents the sum of the extra cost (higher cost - minimum cost of a request) incurred by satisfying some

TABLE II
TOTAL AND EXTRA COST FOR SATISFYING REQUESTS WITH TOP k TEMPLATES

Top k	Total cost ($)	Extra cost as a percentage of total cost
5	341,648.20	3.6
10	933,924.80	10.6
15	882,220.50	5.4
20	1,541,252.80	19.0
25	1,413,980.00	8.4
89	1,676,451.00	0.0

server requirements using the top k templates. The extra cost is shown as a percentage of the total cost.

The results summarized in Table II enable a cloud provider to do quantitative and systematic trade-off analysis between the number of templates and the cost gain/loss. For example, consider the comparison of top 10 and top 15 templates (rows 2 and 3 of Table II). We observe that increasing the number of templates from 10 to 15 yields only a small (5%) reduction in extra cost, which may not be a significant savings when compared to the extra management overhead of the 5 additional templates. Regarding execution times, CloudAffinity is able to generate template lists in a couple of minutes and would be practically applied in industry on a one-time basis when assessing customer workloads.

VI. RELATED WORK

Matching applications' server requirements to available resources has long been investigated [4], [5]. Virtualization has simplified this task by allowing users and system administrators to create VM instances and images that encapsulate the whole software stack required by applications to run [6]. Repositories and catalogs for VM images have been proposed [7], [8] with the main goal of storing images and allowing users and systems to quickly retrieve them during service deployment. As discussed beforehand, several issues arise as the number of virtual resource templates grows: (i) the time for users to select a template increases; and (ii) patching images can become a challenge [9], [10].

These issues have been addressed in different ways. The Open Virtualization Format (OVF) attempts to provide a hypervisor-neutral format for images that can simplify their packaging, distribution and management [11]. Filepp *et al.* [3] tackle the problem of choosing an image for a target server by selecting the image that yields the smallest software migration cost (*i.e.* the cost for installing the required software and removing software that is not necessary). SOAVM [12] provides a framework for automatic software configuration (*i.e.* downloading and installation). Moreover, it maintains basic images that are customised with the proper software when VMs are instantiated.

Existing work has also proposed enhancements to current repositories by provisioning semantic descriptions of the contents of images [1]. Mirage [8] eases image capture and

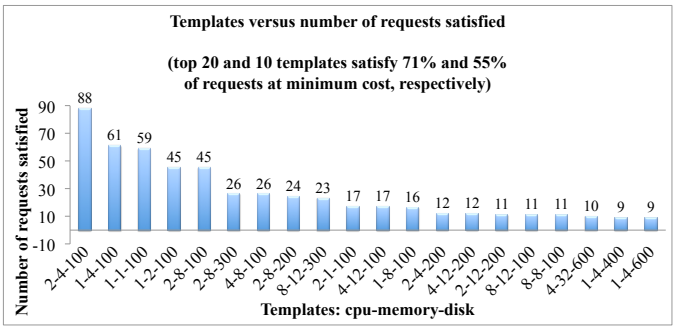


Fig. 8. Templates and the number of server requirements satisfied by each of them. Data for the most used (top) 20 templates are shown

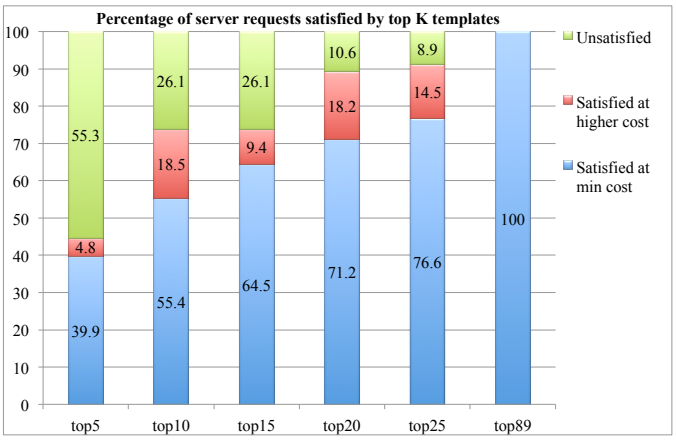


Fig. 9. Each bar shows for a given top k templates, what percentage of the 747 requests are satisfied at minimum cost, higher cost and unsatisfied. top89 represents the set of all templates needed to satisfy all the server requirements at minimum cost.

deployment by storing them in a format that allows them to be compared with one another, also attempting to provide semantic search on the available images [13]. In addition, it maintains a tree that describes how an image has derived from other images. There are also attempts to minimize the time for deploying VM images on their respective hosts through several techniques such as delta deployments [8] and VM image caches [14].

The cost of borrowing resources from a cloud provider to serve an organization's or Grid's demands using their proposed templates has been investigated [15], [16], [17]. Moreover, Mao *et al.* [18] point out the challenge of selecting virtual resource templates and propose techniques for scaling an application on a cloud. Dastjerdi *et al.* [19] propose a service that matches OVF instances provided by users to appropriate offerings of cloud providers. However, to the best of our knowledge the previous work does not focus on identifying the set of virtual resource templates that satisfy the largest number of user requests.

VII. Concluding Remarks

This paper evaluated the resource waste caused by limited VM templates and analysed how well a workload of a real data center would fit into the current set of Amazon EC2 templates. In addition, it proposed a framework, termed as CloudAffinity, which considers template costs, and analysed the cost-benefits of having a few sets of templates that can satisfied a large group of VM user requests.

We showed that standard methods, such as K-means clustering, cannot be directly applied to select templates when considering a real data center workload, where users have various and unrelated hardware requirements. An interesting result is that 89 templates are required to meet the server requirements of 747 VM requests, hence showing that few templates (*i.e.* small, medium, and large) may not be able to satisfy a considerable number of requests. However, the results also show that a group of 10 templates can meet most of the requests, and by increasing the number of templates after 10, the cost benefit is minimum, and represents a low advantage for most customers. Therefore, the framework and methods presented in this paper can assist cloud providers to better analyse and determine the optimal set of templates required to meet user demands. As a next research step, we will incorporate in the CloudAffinity framework an online refinement of the set of templates based on updates of customer demands. We will also investigate improvements in resource allocation enabled by CloudAffinity.

References

[1] D. Reimer, A. Thomas, G. Ammons, T. Mummert, B. Alpern, and V. Bala, "Opening black boxes: Using semantic information to combat virtual machine image sprawl," in *ACM SIGPLAN/SIGOPS VEE*, 2008, pp. 111–120.

[2] G. A. et al., "RC2 - A Licing Lab for Cloud Computing," IBM Research Report, Feb. 2010.

[3] R. Filepp, L. Shwartz, C. Ward, R. D. Kearney, K. Cheng, C. C. Young, and Y. Ghosheh, "Image selection as a service for cloud computing environments," in *IEEE SOCA*, Perth, Australia, Dec. 2010, pp. 1–8.

[4] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor – a hunter of idle workstations," in *8th International Conference of Distributed Computing Systems*, San Jose, USA, Jun. 1988, pp. 104–111.

[5] M. Siddiqui, A. Villazón, and T. Fahringer, "Grid Capacity Planning with Negotiation-Based Advance Reservation for Optimized QoS," in *ACM/IEEE Supercomputing 2006*, 2006, p. 103.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SOSP 2003*, New York, USA, 2003, pp. 164–177.

[7] S. Ostermann, R. Prodan, and T. Fahringer, "Extending grids with cloud resource management for scientific computing," in *IEEE/ACM Grid*, Banff, Canada, Oct. 2009, pp. 42–49.

[8] G. Ammons, V. Bala, T. Mummert, D. Reimer, and X. Zhang, "Virtual machine images as structured data: the mirage image library," USENIX HotCloud 2011, Jun. 2011.

[9] T. Garfinkel and M. Rosenblum, "When virtual is harder than real: Security challenges in virtual machine based computing environments," in *10th USENIX HotOS*, Berkeley, USA, 2005, pp. 20–20.

[10] W. Zhou, P. Ning, X. Zhang, G. Ammons, R. Wang, and V. Bala, "Always up-to-date: Scalable offline patching of vm images in a compute cloud," in *ACSAC 2010*, New York, NY, USA, 2010, pp. 377–386.

[11] "Open virtualization format white paper," DMTF, Jun. 2009.

[12] Z. Cheng, Z. Du, Y. Chen, and X. Wang, "SOAVM: A service-oriented virtualization management system with automated configuration," in *IEEE SOSE '08*, Jhongli, Taiwan, Dec. 2008, pp. 251–256.

[13] M. Satyanarayanan, W. Richter, G. Ammons, J. Harkes, and A. Goode, "The case for content search of VM clouds," in *Computer Software and Applications Conference Workshops*, Los Alamitos, USA, 2010, pp. 382–387.

[14] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *HPDC 2008*, New York, USA, 2008, pp. 87–96.

[15] M. D. de Assunção, A. di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using Cloud computing to extend the capacity of clusters," in *HPDC 2009*, 2009, pp. 141–150.

[16] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the Cloud: The montage example," in *ACM/IEEE Supercomputing*, 2008, pp. 1–12.

[17] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for science Grids: a viable solution?" in *DADC'08*, 2008, pp. 55–64.

[18] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *IEEE/ACM Grid*, 2010, pp. 41–48.

[19] A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya, "An effective architecture for automated appliance management system applying ontology-based cloud discovery," in *IEEE/ACM CCGrid*, Melbourne, Australia, May 2010, pp. 104–112.