RC25417 (WAT1310-030) October 10, 2013
Computer Science

IBM Research Report

A Predictive Method for Identifying Optimum Cloud Availability Zones

Merve Unuvar, Yurdaer Doganata, Malgorzata Steinder, Asser Tantawi
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA

Stefania Tosi
University of Modena and Reggio Emilia
Modena, Italy

Research Division
Almaden – Austin – Beijing – Cambridge – Dublin - Haifa – India – Melbourne - T.J. Watson – Tokyo - Zurich

A predictive method for identifying optimum cloud availability zones

Merve Unuvar, Yurdaer Doganata, Malgorzata Steinder, Asser Tantawi
IBM T. J. Watson Research Center
Yorktown Heights, NY
munuvar, yurdaer, steinder, tantawi@us.ibm.com

Stefania Tosi
University of Modena and Reggio Emilia
Modena, Italy
stefania.tosi@unimore.it

***Abstract*—Cloud service providers enable enterprises with the ability to place their business applications into availability zones across multiple locations worldwide. While this capability helps achieve higher availability with smaller failure rates, business applications deployed across these independent zones may experience different Quality of Service (QoS) due to heterogeneous physical infrastructures. Since the perceived QoS against specific requirements are not usually advertised by cloud providers, selecting an availability zone that would best satisfy the user requirements is a challenge. In this paper, we introduce a predictive approach to identify the cloud availability zone that maximizes satisfaction of an incoming request against a set of requirements. The predictive models are built from historical usage data for each availability zone and are updated as the nature of the zones and requests change. Simulation results show that our method successfully predicts the unpublished zone behavior from historical data and identifies the availability zone that maximizes user satisfaction against specific requirements.**

Keywords*—*Availability zones, multiple data centers, cloud, predictive analytics, performance analysis.

I. Introduction

A user of an infrastructure cloud service such as EC2 has the option of selecting an availability zone where their resources are provisioned. An availability zone (AZ) is a data center that is physically isolated from other availability zones. Cloud providers offer several of such availability zones in various geographies. For any cloud provider, availability zones are not identical - their hardware infrastructure type, version of management stack, and load characteristics differ. As a result, services offered by different availability zones will vary.

Some differences in services offered by different availability zones are widely known as they are published by the service provider. Such well known attributes are types of offered instances, their sizes, and prices. As an example, as of writing this paper, EC2 does not offer *Cluster GPU* instances in Sao Paulo zone while they are available in North Virginia zone. However, cloud users have observed a number of differences in the quality of service provided by availability zones that are not captured in those advertised attributes.

Benchmarking studies performed on EC2 report on variation between performance of same type instances deployed in different availability zones. Shad et al. [1] measured significant differences in instance disk read/write performance, as well as in instance network performance between availability zones in US and EU locations. They conjecture that divergent disk performance is due to different HDD technologies used in US and EU. They also stipulate that differences in network performance result from different virtual machine placement algorithms applied in different availability zones. Furthermore, *Life Scaling* blog[1] reports up to 100% difference in network latency between pairs of instances deployed within different availability zones in the US East region. Farley et al. [2] and Ou et al. [3] measured substantial performance variation (up to 280%) between same-type instances deployed on different processor architectures in EC2 with benchmarks stressing various types of resources. Furthermore, Ou et al. [3] estimated up to 9.5 times higher probability of acquiring an instance on a specific processor architecture between two availability zones indicating that, due to infrastructure differences, the expected instance performance in these two zones should vary considerably. Outside of scientific studies, users of EC2 report particularly high probability of instance deployment rejection due to resource contention in certain availability zones (mostly in the US East region).

The above data suggest that considerable difference in quality of service may be observed by cloud users with different availability zones, even when offered by the same provider. This observation is not surprising as different data centers are likely made of different generations of hardware and software technologies. Moreover it should be expected that those characteristics will change over time. From a user perspective, such inconsistency in service availability and behavior is an unwelcome challenge as availability zone selection may have a rather drastic impact on the availability, performance, and cost-efficiency of the deployed workload. A tool that would help a user to automatically select an availability zone suitable for their requirements would alleviate these concerns. As the QoS attributes needed to make a decision are not usually advertised as part of a service description, in addition to the fact that they may vary over time, a suitable tool will have to estimate such attributes based on prior observations.

In this paper, we present an adaptive and predictive method for automatically selecting availability zones. In the proposed model a user provides a set of desired quality of service attributes for each instance deployment request and their importance. Based on this input, we construct a utility function. QoS measurements for previously deployed instances by the same or other users are available with their associated utility value. We use these observed utility values to train prediction models for each availability zone. In essence, the prediction

[1] http://orensol.com/2009/05/24/network-latency-inside-and-across-amazon-ec2-availability-zones/

models help us learn unpublished attributes of a service. We accommodate time varying changes in service attributes by reconstructing the models based on continuously changing input data. Given the prediction model and the requirement specified in a new request, we formulate and solve an optimization problem to select the optimum availability zone for that request.

The proposed solution moves cloud service users from the common practice of manually selecting a cloud availability zone, by relying on community knowledge, to the automatic selection of customized solutions, focused on their needs. It is important to note that our solution considers the possibility of a deployment across multiple providers, i.e., the resources can be placed in different zones of different providers if such a deployment is supported by other important factors such as availability of suitable images and automation, similarity of service types, etc. In a multi-cloud setting our solution would benefit cloud brokering services which become increasingly popular.

The remainder of this paper is structured as follows. Section II describes the problem formulation and details the main elements of our model construction. Section IV presents the configuration of the simulated environment used for extracting performance results, which are presented in Section V. Section VI surveys related works. And, Section VII concludes the paper with final remarks and future work.

II. Problem Definition

We consider the system depicted in Figure 1. In this system, we assume either the availability zone manager, or a monitoring tool deployed within it, provide the necessary measurements to evaluate the *satisfaction level* of each requirement specified in the request. Information about architectural features of a node, failure and recovery notifications, runtime performances such as throughput of various resources, latency, etc. can be gathered by monitoring. To obtain such a measurement, one can use a monitoring service provided by the cloud owner, e.g., Cloud Watch offered by EC2, or deploy a proprietary tool to monitor the deployment and runtime characteristics of provisioned instances.

Figure 1 shows the monitoring agent that collects measurements and evaluates how much the requirements specified in a request are satisfied. The evaluation results are then passed to the cloud manager. In this paper, we focus on the cloud manager part of the system, where we build prediction models to decide which availability zone satisfies the user's requirement most. The only input that is needed for cloud manager from the monitoring agent is the requirement *satisfaction vector*. With the input of user requirement and their associated weights, cloud manager makes a decision on the availability zone that meets the user requirement most.

A. Formulation of the problem

In this Section, we describe the user request attributes, details of the availability zone model and the associated measures.

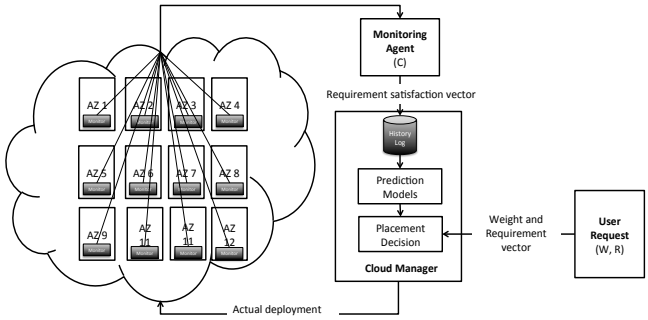

Fig. 1. System diagram

1) Request attributes: A user request is represented by a *requirement vector*. Let the i^{th} user request be represented by the vector, $\boldsymbol{r_i} = [r_{i1}, r_{i2}, \ldots, r_{iJ}]$, where r_{ij}, $j = 1, \ldots, J$, specifies the j^{th} requirement of user *i* that is expected to be satisfied by the selected availability zone. User requirements may include: (i) *resources*, as the resource amounts required by the user (e.g., CPU, memory etc); (ii) *QoS criteria*, as quality of service objective that a user wants to achieve (e.g., highest reliability, minimum latency, highest performance); (iii) *constraints*, as restrictions on possible service provisioning (e.g., locality, throughput, load balancing constraints); (iv) *user instance types*, as the type of instance the user wants to run; and (v) *user machine types*, as the type of machine that the user requires the availability zone to provide.

2) Availability Zone Model: Availability zones differ in characteristics depending on their implementation. Attributes such as availability zone size, hardware infrastructure, or management stack (including instance placement policies) result in different levels of reliability and performance. Attribute values that influence the QoS offered by an availability zone for a particular instance type are not known. Generally, QoS data for any particular instance type in a given availability zone are not known a priori. It is, however, possible to measure the QoS parameters after an instance has been deployed. Such measurements may be evaluated against the requirements specified in the request. We call the results of such an evaluation a *requirement satisfaction level*.

Let $c_{ij} \in [0, 1]$ denote the *satisfaction level* of requirement r_{ij}. If the requirement r_{ij} is fully satisfied, then $c_{ij} = 1$, otherwise $0 \leq c_{ij} < 1$. In this paper, we are not prescribing a method to measure how much a user requirement is satisfied after deployment. Rather, we are stating that such an evaluation can be done using any existing method to produce the vector of *satisfaction levels*, $\mathbf{C}_i^T = [c_{i1}, c_{i2}, \ldots, c_{iJ}]$, for each incoming request $\boldsymbol{r_i}$, deployed to an availability zone. The input $\mathbf{C}_i^T$ is the only input that cloud manager needs from the monitoring agent.

III. Predictive Method

In this Section, we explain how to build a prediction model [4], by learning the behavior of each availability zone based on the historical data of past requests. Our prediction model maps the *requirement vector* of an incoming request, $\mathbf{r}_i = [r_{i1}, r_{i2}, \ldots, r_{iJ}]$, to a customer satisfaction measure defined by a utility function, $f(\mathbf{r}_i) \in [0, 1]$. The utility

function reaches its maximum value of one when there is complete satisfaction. The value of $f(\mathbf{r}_i)$ depends on how much the requirements of an incoming request is satisfied by the availability zone where the request is placed. The vector of *satisfaction levels*, $\mathbf{C}_i^T = [c_{i1}, c_{i2}, \ldots, c_{iJ}]$, for each incoming request $\mathbf{r}_i$, is observed after the request is deployed. The satisfaction of some requirements may be more crucial than others, therefore the *satisfaction level* of each requirement may have different significance. The *weight vector* $\mathbf{W}_i^T = [w_{i1}, w_{i2}, \ldots, w_{iJ}]$ denotes the *significance levels* for requirement attributes for request $\mathbf{r}_i$. The more the value of w_{ij}, the stronger the significance of j^{th} requirement is. One possible way of defining the utility function $f(\mathbf{r}_i)$ is to take a linear combination of the *satisfaction level* for each incoming request $\mathbf{C}_i$ and the associated weights $\mathbf{W}_i^T$ multiplied by an indicator function $\phi(\mathbf{r}_i)$. The indicator function is used to set the *satisfaction level* to zero when the request is rejected. Hence, we define the utility function as,

$$f(\mathbf{r}_i) = \phi(\mathbf{r}_i) \sum_{j=1}^{J} w_{ij} c_{ij} = \phi(\mathbf{r}_i) \mathbf{W}_i{}^T \mathbf{C}_i, \tag{1}$$

where

$$\phi(\mathbf{r}_i) = \begin{cases} 0, & \text{if the request is rejected,} \\ (\sum_j w_{ij})^{-1}, & \text{otherwise.} \end{cases} \tag{2}$$

Note that the selection of $\phi(\mathbf{r}_i) = (\sum_j w_{ij})^{-1}$, in the case of no rejection, normalizes the *weight vector* and limits the maximum possible value of $f(\mathbf{r}_i)$ to 1.

A. Example

Let $\mathbf{r}_i$ =$[r_S, r_L, r_I, r_A, r_{Mh}]$ be the *requirement vector* of an incoming request. The request contains requirements related to the size, supported instance type, infrastructure type, and reliability of an availability zone. The description of the requirement attributes are listed below.

- r_S: Requested CPU and RAM resources where $r_S \in$ {*micro, small, medium, large, xlarge*}
- r_L: Level of reliability where $r_L \in$ {*low, medium, high*}
- r_I: Tolerance to interruption where $r_I \in$ [*0, 1*]
- r_A: Requested instance type where $r_A \in$ {*compute intense, storage, memory intense Instance* }
- r_{Mh}: Requested machine type where $r_M \in$ {*Intel Xeon-series, AMD Opteron-series*}

The measured satisfaction for each requirement is captured by vector $\mathbf{C}_i^T$. Let's assume that for an incoming request with a *requirement vector* $\mathbf{r}_i$ = ["large", "medium", "1", "compute intense","Intel Xeon-series"], the *satisfaction vector* is observed as; $\mathbf{C}_i^T = [0, 1, 0, 1, 1]$. Note that partial *satisfaction levels* are not considered to simplify the example. This means that the size and tolerance to interruption requirements of the incoming request, $r_S = \{\text{"}large\text{"}\}$ and $r_I = \{\text{"1"}\}$, are not satisfied while other requirements are fully satisfied. If the associated *weight vector* is $\mathbf{W}_i^T = [0.2, 0.3, 0.3, 0.1, 0.1]$, then the utility function for $\mathbf{r}_i$ is computed as,

$$f(\mathbf{r}_i) = \begin{cases} \phi(\mathbf{r}_i) \mathbf{W}_i{}^T \mathbf{C}_i = 0.5, & \text{if request is placed,} \\ 0, & \text{if request is rejected,} \end{cases}$$

where $\phi(\mathbf{r}_i) = 1$ if the request is placed and 0 otherwise. Note that due to the weights associated with each requirement, the *satisfaction level* did not exceed 0.5 when more than half of the requirements are satisfied.

B. Predicting the utility function

If there are multiple availability zones providing the same services, it is natural that the availability zone which returns the maximum utility value $f(\mathbf{r}_i)$ for the incoming request is likely to satisfy the requirements most. The exact value of the utility function can be computed only after deployment. It can be, however, predicted based on the utility values of deployed requests in the past. In this paper, we propose to build a prediction model for every availability zone and use the predicted utility values, $\tilde{f}(\mathbf{r}_i)$, in selecting an optimum availability zone.

Figure 2 depicts how we create the training tables and the associated prediction models for each availability zone. First, the incoming requests are placed into the availability zones by using a random selector. This way the incoming requests are distributed uniformly to each zone. After random placements of requests, the associated *satisfaction level* vectors are computed and stored in a history log (Figure 2(a)). Then for an incoming request $\mathbf{r}$, a training table is built for every zone based on its *weight vector* $\mathbf{W}$ and the history of *satisfaction vectors* of random arrivals to each availability zone from the history log (Figure 2(b)). Once the training tables are built for each zone, the associated prediction models are generated by using machine learning techniques for the incoming request. In short, prediction models are built based on the history log and the *weight vector* of an incoming request for every zone at the time of arrival. Let $\mathbb{P}_n$ denote the prediction model for the

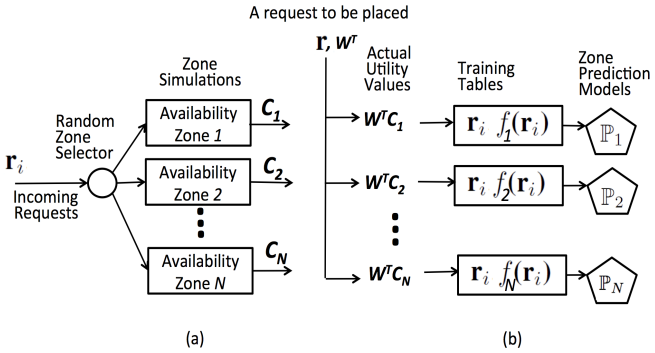

Fig. 2. (a) Collecting satisfaction level history (b) Generating predictive model for each arrival

satisfaction level of an incoming request placed in an availability zone n. $\mathbb{P}_n$ is trained by sample records or instances characterized by the tuple $(\mathbf{r}_i,\ f_n(\mathbf{r}_i))$ for $i = 1, \ldots, I$, where I is the size of the training set, or the number of the instances used for training. Here, $f_n(\mathbf{r}_i)$ is the empirical value of the utility function in zone n, associated with the *requirement vector* $\mathbf{r}_i$ in the history log and new coming request $\mathbf{r}$'s weight vector $\mathbf{W}^T$, hence we bias prior requests' satisfaction levels with the incoming request weight in the training table. Classification models assume that utility values are discrete. In the case that the utility value is continuous, regression models should be used for prediction. Table I depicts the structure

of the training data set used by $\mathbb{P}_n$ to learn the behavior of availability zone n. After the training phase, $\mathbb{P}_n$ learns how to

TABLE I. TRAINING DATA SET FOR ZONE n.

Case	Attributes				Target
1	r_{11}	r_{12}	...	r_{1J}	$f_n(\mathbf{r}_1)$
2	r_{21}	r_{22}	...	r_{2J}	$f_n(\mathbf{r}_2)$
3	r_{31}	r_{32}	...	r_{3J}	$f_n(\mathbf{r}_3)$
.	.	.	...	.	.
I	r_{I1}	r_{I2}	...	r_{IJ}	$f_n(\mathbf{r}_I)$

predict the *satisfaction level* for the *requirement vector* $\mathbf{r}$ as given by,

$$\mathbb{P}_n(\mathbf{r}) = \tilde{f}_n(\mathbf{r}). \tag{3}$$

Here $\tilde{f}_n(\mathbf{r})$ is the predicted *satisfaction level* for zone n. The estimated mean squared prediction error, $\bar{e}(\mathbb{P}_n)$ for model $\mathbb{P}_n$, is given by

$$\bar{e}(\mathbb{P}_n) = \frac{1}{I+m} \sum_{\ell=1}^{I+m} [f_n(\mathbf{r}_\ell) - \tilde{f}_n(\mathbf{r}_\ell)]^2. \tag{4}$$

where m is the number of requests that are deployed to a zone after prediction. In order to find an unbiased estimate of the predicted error, the trained model is validated by testing it against a set of *requirement vectors* that are not part of the training set. Cross-validation [5] is a technique often used in machine learning to evaluate the models by dividing the sample data into training and validation segments. The first segment is used to learn the model and the second one is to validate. Equation 4 shows how to estimate the testing error by using I test data. In typical cross validation, the training and validation sets must cross-over in successive rounds such that each data point has a chance of being validated. We use k-fold cross validation by randomly partitioning the sample into k sub-samples where a single sample is retained as the validation sample and $k-1$ sub-samples are used as training data to measure the accuracy of our prediction models.

Once the prediction models, $\mathbb{P}_n$s, are generated for each zone then they are employed to select the availability zone that maximizes the satisfaction of an incoming request as described in the next Section.

C. Selecting an availability zone

Had the value of the utility function been known before deployment, the availability zone that satisfies the customer requirements most would have been an obvious choice. The utility values, however, strongly depend on unpublished zone properties and they are not publicly available. Regardless, we can predict them by using the machine learning techniques and the historical data as explained above and use predicted utility values for availability zone selection. One possible selection policy uses the maximum predicted utility value to select the availability zone. Hence, if there are n availability zones, the zone that satisfies the *requirement vector* of the incoming request $\mathbf{r}$ most is represented with the equation,

$$\mathbb{P}_S(\mathbf{r}) = \max_n \{\mathbb{P}_n(\mathbf{r})\} \quad \forall n \leq N. \tag{5}$$

Here the utility function is maximized when $n = S$. Therefore, the zone selector assigns zone S as the optimal zone for the incoming request $\mathbf{r}$ provided that zone S has enough capacity. If the best zone that maximizes the utility function cannot accommodate the associated request due to shortage in zone capacity, then the second best zone is selected for placing the incoming request. This procedure repeats until the request is placed (or finally rejected). The left hand side of Figure 3 shows how the zone selection procedure loops until the best available zone that can accommodate the request is determined.

D. Updating the prediction models

Due to the dynamic nature of the cloud environment (i.e. availability, load, time etc.), the accuracy of the prediction models may decrease over time. Decline in prediction accuracy happens when the availability zone features change significantly. These changes may not be publicized or become available to the placement policy. Therefore, as the zone features change, the prediction models need to be retrained to learn the new zone behavior. We monitor the prediction error by using Equation 4 and run a k-fold cross validation after a number of new requests are placed. After each placement, the training data set in the history log is updated. The prediction models are retrained when the average prediction error increases above a preset threshold.

The right hand side of Figure 3 shows how the new training samples are collected dynamically to retrain the prediction models when the prediction error exceeds the threshold value.

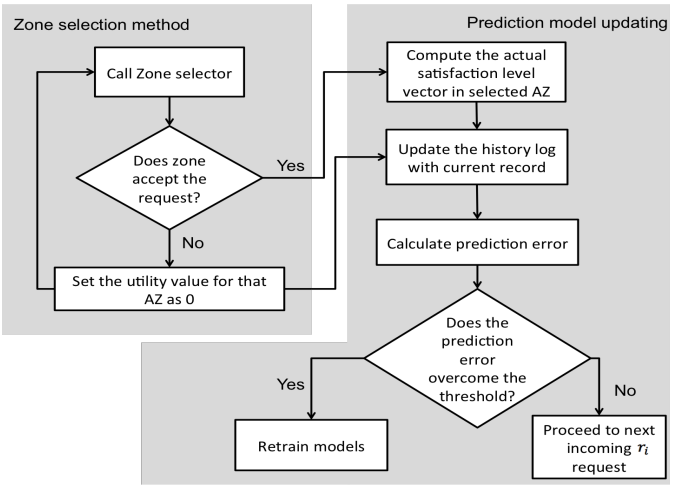

Fig. 3. Zone selection method and updating prediction models.

IV. DESCRIPTION OF SETUP

In this Section, we describe the simulation environment and experimental setting that we developed for evaluating the predictive method for selecting cloud availability zones.

A. Simulation environment

We used a Java based simulation environment to simulate various availability zones with different characteristics and an inter-arrival process for incoming heterogeneous user requests. Figure 4 shows the components of our simulation infrastructure. It has mainly three components. The first component is the *zone simulator* which simulates availability zones with various CPU and memory capacities, reliability levels (by injecting different failure types with different probabilities), internal

placement algorithms, delays, supported instance and/or infrastructure types. The second component is the *arrival process simulator* where incoming requests with various requirements are simulated based on specified inter-arrival and lifetime distributions. The desired load factor for an availability zone is controlled by the parameters of the inter-arrival distribution in the arrival process simulator. The third component is the *zone selection* layer which uses predicted utility values to select the availability zone that would best satisfy an incoming request. After the zone models are generated, by using the machine learning techniques as explained in Section III, the utility value for an incoming requests is predicted for each zone separately.

After every incoming request generated by the *arrival process* is placed into an availability zone by the *zone selection* layer, the *zone simulator* checks whether or not the requirements are satisfied. Hence, the implemented satisfaction vector is a binary vector. The request requirements and the associated utility value constitute a training sample for the zone that the request is placed into. In order to generate the training data, the same simulation infrastructure is used except that the zone selector and the prediction models are replaced by a random zone selector as depicted in Figure 2. After generating a sufficient number of training data sets for each zone, the training data sets are passed to IBM SPSS Modeler [6], which offers a variety of modeling methods to develop prediction models for the utility value of an incoming request. We generate and compare several prediction models for each zone. The prediction models we consider include Neural Networks, Decision Trees, and Support Vector Machines. Among various models, we choose the one with the highest accuracy.

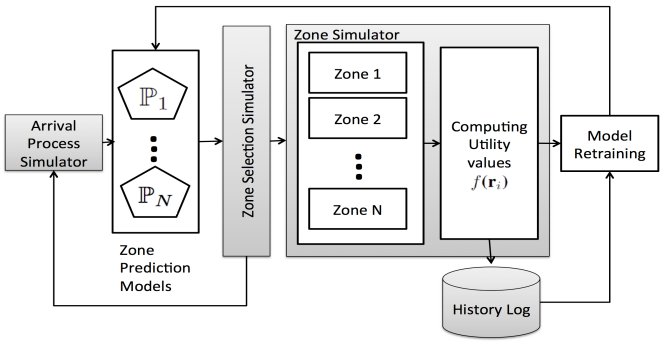

Fig. 4. Simulation environment.

One other aspect of the *zone simulator* is that it compares the predicted utility value with the actual utility value observed after the placement. If the difference between actual and predicted values increases, the initial prediction accuracy is reduced. In this case, the models are trained again with the latest data stored in the history log.

B. Experimental setting

1) Simulated zone attributes: We simulated 12 availability zones in various locations worldwide with different capabilities that are listed in Table II. Most of these attributes, especially the ones related to performance, are not published and known a priori (e.g., Reliability levels, placement algorithms). Each availability zone is composed of 10 racks, each rack contains 2 blade centers and each blade center contains 50 physical machines (PMs). An availability zone is simulated to support a heterogenous hardware infrastructure, composed of a mix of processor models. The used CPU models are: Intel Xeon series "E5507", "E5430", "E5645", "X5500", "X5570" and AMD Opteron series "2218HE", and "270". We define 3 different heterogenous infrastructure types depending on the percentage of hardware mix in a zone. Table III shows the exact percentage of CPU model mixes per infrastructure type.

TABLE II. CLOUD AVAILABILITY ZONES ATTRIBUTES.

#AZ	Region	Reliability level	Placement algorithm	Support for high performance instance(s)	Hardware type
1	US- East 1	low	random	High Storage	I
2	US- East 2	low	random	High Compute	I
3	US- East 3	low	random	High Memory Speed	I
4	US- West 1	medium	random	High Storage	II
5	US- West 2	medium	random	High Compute	II
6	EU 1	medium	random	High Memory Speed	II
7	EU 2	medium	LeastFull	High Storage	III
8	South America 1	medium	LeastFull	High Compute	III
9	South America 2	medium	LeastFull	High Memory Speed	III
10	Asia Pacific 1	high	LeastFull	High Network I/O + SSD	I
11	Asia Pacific 2	high	LeastFull	High Network I/O + SSD	II
12	Asia Pacific 3	high	LeastFull	High Network I/O + SSD	III

TABLE III. HARDWARE CONFIGURATION TYPES.

Infrastructure type	Processor brand	CPU model	% of CPU models
Type I	Intel Xeon	E5507	50%
	Intel Xeon	E5430	30%
	Intel Xeon	E5645	20%
Type II	AMD Opteron	2218HE	70%
	AMD Opteron	270	30%
Type III	Intel Xeon	X5500	60%
	Intel Xeon	X5570	40%

The monitoring service provided by the cloud owner monitors the available capacity in all zones. Incoming requests are rejected in a zone only if there is not enough capacity to accommodate the request in the targeted PM for placement. For the sake of simplicity, we simulated each zone with the same CPU and memory capacity with 32 cores and 32 GB memory, respectively. In order to create performance variations, we introduce two types of internal placement algorithms: random and load balancing. For availability zones 1-6, a random placement algorithm where the instances are placed randomly to PMs is implemented. For the rest, a placement algorithm which balances the load by selecting the least full PM is implemented. As for the unpublished characteristics of the zones, each availability zone is simulated to support different high performance instance types. For instance, high network instances with bandwidth requirement of 4KB perform best in Asia Pacific zones (see Table II). Even though instance types are advertised as a selection to user, their performance per zone is not published by the cloud provider. The failures are simulated such that either entire rack or entire blade center fails with either low or high probabilities. Depending on the failure probability, blade center/rack failure and availability information, reliability levels are classified as low, medium and high. The mean time between failures is Gamma distributed, while lifetime of failures (i.e., time to recovery) follows a LogNormal distribution. For low reliability zones, we inject both rack and blade center level failures with Gamma(1, 0.33), for medium reliability zones we inject rack level failures with Gamma(0.75, 0.2) and for high reliability zones we inject only blade center failures with Gamma(3, 1) where parameters are

in hours. All three zones have LogNormal(2,0.3) distribution for time to recovery where parameters are in minutes.

2) Simulated request attributes: We generated training data sets by simulating multiple instances of user requests with different attributes. As in the example reported in Section II, the *requirement vector* of an incoming request has the form $\mathbf{r}_i = [r_S, r_L, r_l, r_A, r_{Mh}]$. Each attribute in the vector takes values among those listed in Table IV.

TABLE IV. USER REQUEST ATTRIBUTES.

r_S	r_L	r_I	r_A	r_{Mh}
micro	low	[0,1]	High Storage	Intel Xeon E5507
small	medium		High Compute	Intel Xeon E5430
medium	high		High Memory	Intel Xeon E5645
large			High Network	Intel Xeon X5500
xlarge			SSD	Intel Xeon X5570
				AMD Opteron 270
				AMD Opteron 2218HE

The size of the instances is represented by r_S. The smallest instance size is 1 core with 0.6 GB memory, while the largest instance is 16 cores and 15 GB memory. We simulated 5 different instance sizes where the details of other instance sizes are depicted in Table V. Users also indicate the desired instance type. The types of the instances (e.g., high storage, high compute etc.) that a user can specify are listed in Table IV under the r_A column. Lastly, users indicate their hardware infrastructure preference represented as r_{Mh}. The simulated user hardware choices are listed in Table IV.

As for the zone attributes that are changing over the time, user can also set their preferences up front and the monitoring tool simulated in *zone simulator* in Figure 4 monitors the *satisfaction level* of these requirements over the lifetime of the application. Reliability level is denoted by r_L and users are allowed to specify either low, medium or high reliability zones depending on their application type. Deployed user requests in a zone that has the required or higher level of reliability considered to have full satisfaction ($c_L = 1$) from reliability component of utility function. Users also specify the interruption tolerance that is the sensitivity for being interrupted during their application lifetime. Quantity r_I represents the interruption tolerance level, ranging from 0 to 1. Value 0 is selected for the instances that are very tolerant to interruption, whereas value 1 is selected for the instances that are very sensitive to interruption.

TABLE V. INSTANCE SIZES (r_S).

	$micro$	$small$	$medium$	$large$	$xlarge$
CPU	1	2	4	8	16
Memory (GB)	0.6	1.7	3.5	7.5	15

In addition to *requirement vector*, user sets the weight vector for utility function as , $\mathbf{W}^T = [w_S, w_L, w_I, w_A, w_{Mh}]$. The user weight function of $w_i = 0.2$ for $i \in \{S, L, I, A, Mh\}$ is selected to use normalizing factor of 1.0. As the weight vector indicates, for our experimental settings, all the preferences have equal weights for this user. Our aim is to observe the unbiased impact of each requirement. We use normalizing factor of 1.0 for this particular set up that yields the maximum value of utility function, when all the requirements are satisfied as 1, while the minimum value is 0 when the request is rejected.

3) Request arrival pattern: We simulated all combinations of 5 user attributes (in Table IV) summing up to more than 450 different instance request types with their unique set up. The inter-arrival time of each request type follows an exponential distribution whose parameters depend on the desired load factor (i.e., 95% for our set up), while the request life time follows a uniform distribution. The mix of request types is evenly distributed and this set up is simulated for a 4-week time window.

V. NUMERICAL RESULTS

In this Section, we demonstrate numerical results for different applications of prediction models with the set up described in Section IV.

The placement approach introduced in this paper does not require a priori knowledge of availability zone properties. In other words, the attributes of zones are not available during placement. Hence, placement is made solely based on the predicted behavior of the availability zones which is learned from past behavior. In this Section, we compare the effectiveness of prediction-based placement policy against policies that perform placement by utilizing the state of the availability zone and its properties. Our goal is to show that the impact of availability zone features to user satisfaction can be learned, and that placement policies based on the learned behavior perform better.

In our experiments, we compare our ***Prediction Based*** policy that selects the availability zone with highest utility to other policies making optimal zone selections based on attributes other than utility function. We compare the policy classes listed below.

1) **Random policy**: Incoming requests are placed to availability zones randomly. This policy is used as a baseline. Any non-random policy that considers the characteristics of the zones and the requirements of the incoming requests is expected to perform better than the random policy.
2) **Load-based policies**: Availability zone selection is made merely based on the utilization of the zone at the time of request arrival. If the zone with the highest available capacity is selected for load balancing, we call the policy *Least-Utilized*. On the other hand, if the zone with the lowest availability is selected for lower energy consumption, we call the policy *Most-Utilized*.
3) **Attribute-based policies**: Zone selection is made by matching some of the advertised attributes of the zone with the preferences of incoming requests. In practice, cloud providers publish supported instance types per availability zone. However, they do not publish the performance of the instance types per availability zone. Even though it is not yet practical to publish the zone performances for instance types, we assume that it is hypothetically available to the user. One example of such a policy, *Match Instance Type* policy, makes a selection based on the advertised performance levels of instance types (see Table II) by each zone. If the candidate zone supports the performance level of the instance types required by

the incoming request then it is selected. Similarly, it is not practical for cloud providers to publicize the hardware configuration of their infrastructure hence, the machine types (e.g., CPU models) that instances are running on are not available to the user. For experimental purposes, we assume that such information is available to simulate the case where the machine types are known by the user. The policy, *Match Machine Type*, selects availability zones based on the supported CPU models (see Table III) in different infrastructure types defined in Table II.

Figure 5 reports the average utility value obtained by the different policies for best availability zone selection at different levels of availability zone utilization. The expected utility value for our *Prediction Based* policy is close to 0.85 for all utilization levels. This is significantly higher than the expected utility values achieved by load-based and attribute-based policies. The utility values are less than 0.7 for *Random*, *Least Utilized* and the *Most Utilized* policies, and less than 0.8 for attribute-based policies; *Match Machine Type* and *Match Instance Type*. As expected, *Random* policy achieves the lowest expected utility value. Figure 5 shows that learning the behavior of the availability zones based on historical data satisfies the user requirements better than actually knowing the zone utilization levels and certain zone attributes, such as the type of machines or instances supported.

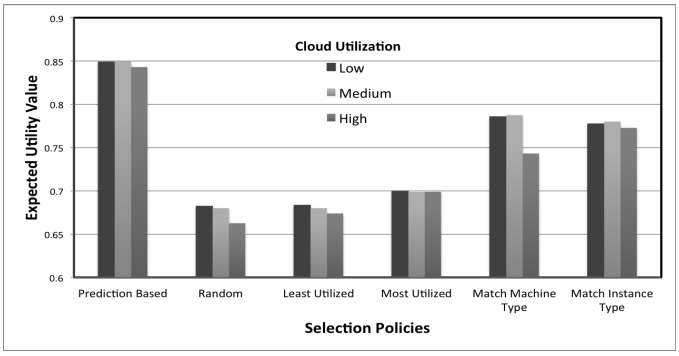

Fig. 5. Comparison of utility values for different policies.

Figure 6 reports the percentage of user requests whose preferred instance types and preferred machine types are supported by the availability zones that they are assigned to. *Match Instance Type* policy satisfies the users application and *Match Machine Type* policy satisfies the users machine type requirements almost all the time. In both cases, the percentage of requests that are matched against their requirements is more than 95%. This is expected, since the *Match Instance Type* and the *Match Machine Type* policies have the prior information about instances and machine types supported by each zone, respectively. Policies other than *Prediction Based* policy match only 30% of the users machine or instance type requirements. As an example, while *Match Instance Type* policy can place to the requests to a zone where the required instance type is satisfied almost 100%, it performs very poor when it comes to matching the machine type requirement. This is because the prior information about the machine types is not available to *Match Instance Type* but it is only available to *Match Machine Type* policy. *Prediction Based* policy, on the other hand, can match both the instance and the machine type requirements significantly higher without any prior knowledge of the distributions of machines and the instance types for availability zones.

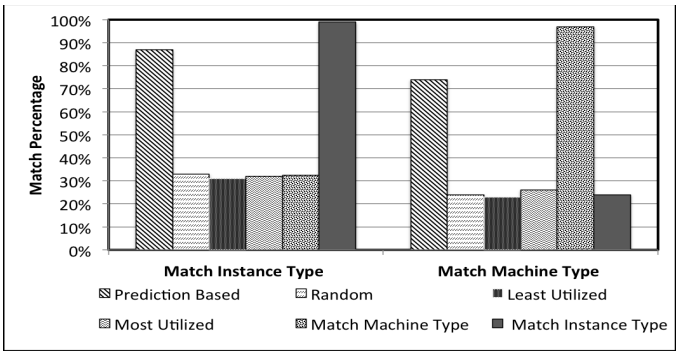

Fig. 6. Percentage of matches in the requirements for different policies.

Prediction Based policy aims to optimize user satisfaction by selecting the zone with the best utility value while maintaining a high resource utilization. In order to measure the impact of the *Prediction Based* policy to resource utilization, we compare the average zone CPU utilization values under different policies. This comparison shows that the *Prediction Based* policy is not reducing the utilization of the zones compared to other policies. Hence, *Prediction based* policy increases the utility value for the user without under utilizing zone resources. The simulation results shows that the average zone utilization for CPU is around 50% for almost all of the policies. The reason for less CPU utilization than offered CPU load is due to high rejection probabilities ($\gtrsim 0.6$) for large and xlarge instances.

In our simulation, we allow every policy to try placing the request multiple times until it is accepted (see Section III-C). Figure 7 reports the average number of trials for each policy in order to get accepted after the first one. We see that the number of additional attempts after the initial one for the *Prediction Based* policy is much less than 1. Therefore, the requests are placed to their best matching zones in their first trials most of the time. In contrast to *Prediction Based* policy, *Most Utilized* and *Match Machine Type* policies require almost 4 additional trials on the average. *Random*, *Least Utilized*, *Match Instance Type*, on the other hand, require more than 2 additional trials on the average. Our *Prediction Based* policy reduces the number of trials significantly compared to other policies. Having less number of trials for placement enables instances to be provisioned faster.

VI. Related Work

To the best of our knowledge, this is the first work to provide a solution to the problem of selecting an availability zone that satisfies the customer requirements most. However, there is an extensive work that has been done in the area of cloud provider selection [7].

In terms of load distribution, an environment is proposed in [8] whereby load is distributed among a number of data centers in a cloud (or multiple vendor clouds) in order to achieve some QoS targets. This results in a federated cloud

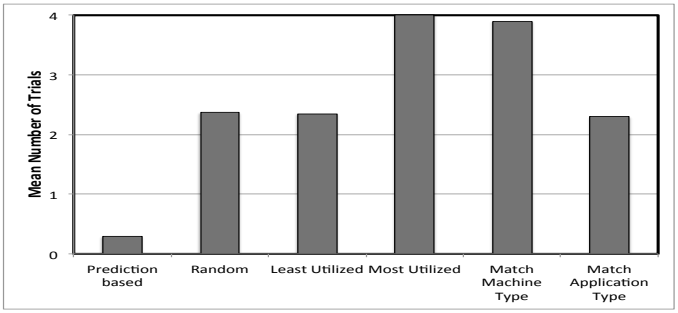

Fig. 7. Average number of trials for different policies.

computing environment (InterCloud) that facilitates just-in-time, opportunistic, and scalable provisioning of application services, supporting dynamic expansion or contraction of capabilities (VMs, services, storage, and database) for handling sudden variations in service demands.

As for selection for the sake of user satisfaction, the comparison depends on the criterion and definition of user satisfaction. Given cost to customers as the satisfaction criterion, [9] describes a splitting algorithm to efficiently split cloud requests among multiple cloud platforms with the objective of decreasing cost. Further, several solutions have been proposed for helping users in cloud selection. Though, they are limited to the optimization of a specific user's objective, such as optimizing costs with the goal of decreasing users investment [10], [11] or reducing risk of service interruptions [12].

The solution approaches to the above-mentioned optimization problems are broadly categorized into two approaches: *static* and *dynamic*. In the static approach [13], it is assumed that the number of required virtual resources is constant and the cloud provider conditions do not change throughout the service life-cycle. Conversely, the dynamic approach is more suitable for variable size services or in the case of changing cloud provider conditions.

Consequently, one needs an efficient selection algorithm that runs repetitively. However, the majority of such solutions [14], [15] use linear programming formulations of the optimization problem, which do not scale when service properties continuously change over time or even impractical when provider's QoS data are unknown.

VII. Conclusion

An automated decision making mechanism for cloud availability zone selection is developed to optimize user satisfaction. Our method employs predictive analytics and machine learning techniques to learn the unpublished attributes of availability zones. The models are dynamically updated to reflect the most recent performance changes in availability zones. Then, the optimum selection is made based on the learned behavior of the availability zones.

We compared our selection method to non-predictive selection methods, which rely on a priori knowledge of unpublished zone attributes. The results show that the predictive approach performs better than the non-predictive ones in terms of user satisfaction, while maintaining high cloud performance.

As future work, we intend to extend our experiments to more complicated workloads, utility functions, and test in a real environment. Another extension of this work, which we wish to pursue, is to investigate the confidence level of the prediction models and its influence on overall decision making results, in terms of user satisfaction and cloud performance.

VIII. Acknowledgments

We would like to thank Giovanni Pacifici for valuable discussions.

References

[1] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 460–471, 2010.

[2] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for your money: exploiting performance heterogeneity in public clouds," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 20:1–20:14.

[3] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, "Exploiting hardware heterogeneity within the same instance type of Amazon EC2," in *USENIX conference on Hot Topics in Cloud Ccomputing*, 2012.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. CRC Press, New York, 1999.

[5] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, vol. 2, no. 12, pp. 1137–1143, 1995.

[6] (2012) IBM SPSS Modeler. [Online]. Available: ibm.com/software/analytics/spss/products/modeler/

[7] Z. ur Rehman, F. Hussain, and O. Hussain, "Towards multi-criteria cloud service selection," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*. IEEE, 2011, pp. 44–48.

[8] R. Buyya, R. Ranjan, and R. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Algorithms and Architectures for Parallel Processing*, ser. Lecture Notes in Computer Science, C.-H. Hsu, L. Yang, J. Park, and S.-S. Yeo, Eds. Springer Berlin Heidelberg, 2010, vol. 6081, pp. 13–31.

[9] I. Houidi, M. Mechtri, W. Louati, and D. Zeghlache, "Cloud service delivery across multiple cloud platforms," in *Services Computing (SCC), 2011 IEEE International Conference on*, 2011, pp. 741–742.

[10] J. L. Lucas Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Dynamic placement of virtual machines for cost optimization in multi-cloud environments," in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*. IEEE, 2011, pp. 1–7.

[11] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*. IEEE, 2009, pp. 103–110.

[12] B. Ofer, E. Hadad, E. K. Kolodmer, D. H. Lorenz, and Y. Moatti, "Optimized placement of virtual machines in a network environment," Jun. 6 2013, uS Patent 20,130,145,368.

[13] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, 2012.

[14] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 228–235.

[15] W. Li, J. Tordsson, and E. Elmroth, "Modeling for dynamic cloud scheduling via migration of virtual machines," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 163–171.