Towards an Autonomic Auto-Scaling Prediction System for Cloud Resource Provisioning

Ali Yadavar Nikravesh, Samuel A. Ajila, Chung-Horng Lung
Department of Systems and Computer Engineering, Carleton University
1125 Colonel By Drive, Ottawa K1S 5B6, Ontario Canada
{alinikravesh, ajila, chung}@sce.carleton.ca

***Abstract*— This paper investigates the accuracy of predictive auto-scaling systems in the Infrastructure as a Service (IaaS) layer of cloud computing. The hypothesis in this research is that prediction accuracy of auto-scaling systems can be increased by choosing an appropriate time-series prediction algorithm based on the performance pattern over time. To prove this hypothesis, an experiment has been conducted to compare the accuracy of time-series prediction algorithms for different performance patterns. In the experiment, workload was considered as the performance metric, and Support Vector Machine (SVM) and Neural Networks (NN) were utilized as time-series prediction techniques. In addition, we used Amazon EC2 as the experimental infrastructure and TPC-W as the benchmark to generate different workload patterns. The results of the experiment show that prediction accuracy of SVM and NN depends on the incoming workload pattern of the system under study. Specifically, the results show that SVM has better prediction accuracy in the environments with periodic and growing workload patterns, while NN outperforms SVM in forecasting unpredicted workload pattern. Based on these experimental results, this paper proposes an architecture for a self-adaptive prediction suite using an autonomic system approach. This suite can choose the most suitable prediction technique based on the performance pattern, which leads to more accurate prediction results.**

Index Terms*— *Autonomic, Auto-scaling, Support Vector Machine, Neural Networks, Workload pattern, Resource provisioning, Cloud computing

I. INTRODUCTION

Cloud computing is an emerging commercial infrastructure paradigm and in the last decade its usage has gained a lot of popularity. The National Institute of Standard and Technology (NIST) specified the essential characteristics of cloud computing, as: On-demand self-service, broad network access, resource pooling, rapid elasticity, and measured services [1]. Elasticity characteristic of cloud computing enables users to acquire and release resources on demand, which reduces their cost by making them pay for the resources they actually have used [2]. Although elasticity is beneficiary in terms of cost, obligation of maintaining Service Level Agreements (SLAs) leads to the necessity in dealing with the cost/performance trade-off.

The three main markets associated with cloud computing include Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). These services can be made accessible to public (public cloud), restricted for private use (private cloud), or be hosted on a hybrid cloud which is a combination of both public and private clouds [3]. This paper addresses the IaaS layer of public cloud computing environments.

Deciding the right amount of resources for a cloud computing environment is a double-edged sword, which may lead to either under-provisioning or over-provisioning [5]. Under-provisioning and over-provisioning are results of, respectively, saturation or waste of resources, and are among the most significant challenges cloud clients are faced with. One approach to overcoming these challenges is to use an *auto-scaling system*. Auto-scaling system solves the cost-performance trade-off by automatically adjusting application's resources based on its workload.

Existing auto-scaling approaches can be classified into reactive, proactive, and predictive algorithms. Reactive algorithms scale in/out system based on current situation of the system. The most significant shortcoming of reactive algorithms in IaaS is neglecting virtual machine (VM) boot-up time, which is reported to be between 5 and 15 minutes [5]. Neglecting VM boot-up time may cause under-provisioning during the lag time, which consequently may lead to SLA penalty. Proactive approaches try to solve this issue by allowing clients to pre-define a schedule where they proactively scale in/out system at certain points of time. Proactive scaling approaches are suitable for the environments with predictable load characteristics. But in the environments with unplanned load spikes employing a predictive auto-scaling system is desirable. Predictive auto-scaling algorithms predict future system behavior and adjust application resources in advance to meet the future needs. Fig. 1 shows an architectural overview of a predictive auto-scaling system.

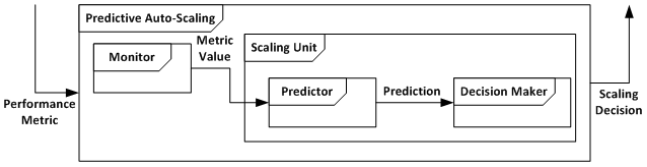

Fig. 1. Architectural overview of a predictive auto-scaling system

As shown in Fig. 1, *Monitor*, *Predictor*, and *Decision Maker* are the main components of a predictive auto-scaling system. To capture current performance of cloud computing

978-0-7695-5567-6/15 $31.00 © 2015 IEEE
DOI 10.1109/SEAMS.2015.22

environment, auto-scaling systems monitor one or more performance metric(s). In this work we considered workload (i.e., the number of user requests per time unit) as the performance metric (see section II.A). As depicted in Fig. 1, *Predictor* uses performance metric's current value from *Monitor* to forecast future performance metric value. The predicted value is sent to the *Decision Maker* which generates the final scaling decision by considering *Cloud Provider Pricing Model*. Since the final scaling decision (decision maker's output) depends on the prediction result, various research studies have focused on improving accuracy of the Predicator's output (see [5] for a comprehensive overview of auto-scaling prediction techniques). According to [5], the most dominant prediction technique in the cloud auto-scaling area is time-series prediction. Time-series prediction technique uses historical performance metric values to forecast future values. Although in the recent years many innovative time-series prediction techniques have been proposed for auto-scaling systems, existing approaches suffer from neglecting the influence of performance metric patterns (i.e., how the metric values change over time) on prediction accuracy. Discovering relations between *performance metric patterns* and *prediction results* can improve prediction accuracy by designing and developing a self-adaptive prediction suite, which chooses the most suitable prediction technique based on the performance metric pattern.

Authors in [5] and [7] report that there are three representative workload patterns in cloud computing environments, with each representing a typical application or scenario. These patterns are: growing, periodic, and unpredicted (see section II.A). Since prediction technique's accuracy varies in accordance to the incoming workload pattern, the hypothesis for this research is:

Prediction accuracy of predictive auto-scaling systems can be increased by choosing an appropriate time-series prediction algorithm based on the incoming workload pattern.

To prove this hypothesis, we have conducted an experiment to compare accuracy of time-series prediction algorithms for different workload patterns. Then, based on the experiment results, we have proposed a high level design for a self-adaptive prediction suite which chooses the most suitable time-series prediction algorithm based on the incoming workload pattern.

In the aforementioned experiment, sliding window technique (see section IV.B) was used to train prediction algorithms. In this technique, window size is one the most important factors that has a significant impact on the prediction accuracy. Thus, to improve prediction results, we have analyzed the influence of sliding window size on prediction accuracy as a part of our experiment.

According to [6], Support Vector Machine (SVM) and Neural Networks (NN) are the most effective prediction algorithms to predict future system characteristics. Therefore, in this work we used SVM and NN algorithms as the *Prediction* component. The main contributions of this work are:

- Comparing SVM and NN prediction accuracy in regards to the different workload patterns
- Analyzing the impact of sliding window size on SVM and NN prediction accuracy
- Proposing a high level design of a self-adaptive prediction suite which chooses the most suitable prediction algorithm based on the incoming workload pattern.

We have used TPC-W web application as the benchmark [8] and Amazon Elastic Compute Cloud (Amazon EC2) as the testing infrastructure. It should be noted that in this paper we focus on the influence of workload patterns on prediction results at the Infrastructure as a Service (IaaS) layer and other IaaS management aspects (such as VM migration and physical allocation of VMs) are out of the scope of this work.

The remainder of this paper is organized as follows: Section II discusses the background and related work. This is followed by mathematical analysis of SVM and NN in section III. Section IV is dedicated to the experiments and results analysis. Section V proposes the high level design of the prediction suite, and finally, conclusion and possible directions for the future research are discussed in section VI.

II. Background and Related Work

In this section basic concepts used in the paper and related work are introduced briefly. Sub-section A is an overview of workload concept and its patterns. In sub-section B, TPC-W and Amazon EC2 are presented, and finally, sub-sections C and D provide an overview of the most dominant auto-scaling approaches in two broad categories: decision making and prediction techniques, respectively.

A. Workload

The term workload refers to a number of user requests, together with the arrival timestamp [4]. Workload is the consequence of users accessing the application or jobs that need to be handled automatically [9]. According to [4] and [6] there are four workload patterns:

- **Stable workload** is characterized by constant number of requests per minute. This means that there is normally no explicit necessity to add or remove processing power, memory or bandwidth for changes in workload.
- **Growing workload** represents a load that rapidly increases.
- **Cycle/Bursting workload** represents regular periods (i.e. seasonal changes) or bursts of load in a punctual date.
- **On-and-off workload** represents the work to be processed periodically or occasionally, such as batch processing.

Authors of [9] categorize application workload types as:

- **Static workload** is characterized by a more-or-less flat utilization profile overtime within certain boundaries.
- **Periodic workload** represents a peaking utilization at reoccurring time intervals experience

- **Once-in-a-lifetime workload** is a special case of periodic workload. In this type of workload the peaks of periodic utilization can occur only once in a very long timeframe. Often, this peak is known in advance as it correlates to a certain event or task.
- **Unpredicted workloads** are generalization of periodic workloads as they require elasticity but are not predictable
- **Continuously changing workloads** represent a constant growth or decrease in application load over time

Resource allocation for batch applications (i.e. on-and-off pattern) is usually referred to as scheduling which involves meeting a certain job execution deadline [4]. Scheduling has been extensively studied in grid environments [4] and also explored in cloud environments, and it is outside of the scope of this paper. Similarly, applications with stable (or static) workload pattern do not require an auto-scaling system for resource allocation per se. Therefore, in this paper we only focus on applications with the following workload patterns:

- **Growing workload:** represents workloads with increasing trend. This class of workload covers growing workload of [4] and continuously changing workload of [9].
- **Periodic workload:** represents workloads with seasonal changes. This class of workload covers cyclic/bursting workload of [4] and periodic and once-in-a-lifetime workloads of [9].
- **Unpredicted workload:** represents fluctuating workloads. This class of workload covers unpredicted workload of [9].

B. TPC-W and Amazon EC2

In order to generate aforementioned workload patterns, one can use either real trace files or application benchmarks. A comprehensive overview of trace files and application benchmarks is reported in [4]. Basically, benchmarks include a web application together with a workload generator that creates session-based requests to the application under test. Some commonly used benchmarks for cloud research are [4]: RUBiS, TPC-W, CloudStone, and WikiBench. In this paper, we have used Java implementation of TPC-W, due to its simplicity and extensive online documentations. Researchers still use TPC-W to conduct auto-scaling experiments [6][17][28]. TPC-W emulates an online bookstore and is commonly used for resource provisioning, scalability and capacity planning for traditional e-commerce websites [5]. The components of TPC-W can be logically divided into three tiers: a set of emulated web browsers, a web server, and a means of persistent storage (i.e., Database tier). The web browsers are emulated using RBE (Remote Browser Emulator) that mimics a number of simultaneous user sessions and allows a single node to emulate several clients.

Besides the benchmark, cloud infrastructure is another important factor that forms experimental setup. In this research work we used Amazon EC2 as our experiment infrastructure. Amazon EC2 is a web-service that provides scalable compute capacity in the cloud and is designed to facilitate web-scale computing for developers [10]. Amazon EC2 offers vast array of instance types as well as open source Application Programming Interface (API), which makes it an ideal infrastructure for our experiment. Amazon EC2 is a central part of Amazon Web Services (AWS) platform and provides developers with a web service, which allows them to boot an Amazon Machine Image (AMI) to create a virtual machine. These virtual machines are called instances and developers are able to create, launch and terminate them as needed. An AMI is a template that contains a software configuration, including an operating system, which defines user's operating environment.

C. Decision Making Techniques

The authors in [4] group existing auto-scaling approaches into five categories: threshold based policies, reinforcement learning, queuing theory, control theory, and time-series analysis. Among these categories, time-series analysis focuses on the prediction side of the resource provisioning task and is not a "decision making" technique per se. In contrast, the threshold-based technique is a pure decision making mechanism while the rest of the auto-scaling categories (control theory, queuing theory, and reinforcement learning) somehow play the *Predicator* and *Decision Maker* roles at the same time.

In contrast to other approaches, threshold-based technique is the only approach, which is widely used in the commercial auto-scaling systems [10-12]. The popularity of this approach is due to its simplicity and intuitive nature. To implement a threshold-based auto-scaling environment, the first step is to monitor one or more performance metrics. As the name suggests, in this technique, the number of VMs varies based on the measures of performance metrics and thresholds, which are set by the operator. There are various threshold-based approaches reported in academic and industrial works. Usually each of these approaches considers four parameters: an upper threshold (thrUp), a lower threshold (thrDown), and two time values vUp and vDown that define how long the condition must be met to trigger a scaling action [4]. Performance of threshold-based technique is highly dependent on these parameters. Hence, setting appropriate values for these parameters is a tricky task. A common problem with auto-scaling, which occurs due to an inappropriate threshold value, is oscillations in the number of VMs. In fact, vUp and vDown parameters are being used to decrease the number of scaling actions and VM oscillations. Some authors have proposed alternative techniques to address this problem. For instance, the work in [13] used a set of four thresholds and two durations. Similarly, some research works (such as [14]) have adopted a combination of rules and a voting system to generate scaling actions.

D. Prediction Techniques

The most dominant prediction technique in the cloud auto-scaling area is time-series analysis [4]. This technique is widely used in various domains such as economics, engineering, and finances [5]. Time-series analysis could be used to find repeating patterns in the workload or to forecast future values.

In order to use time-series analysis in auto-scaling area, a performance metric (such as number of requests per time unit) is periodically sampled at fixed intervals. The result will be a time-series containing a sequence of last observations. Time-series algorithms extrapolate this sequence to predict future value. Some of the techniques used for this purpose in the literature are Moving Average, Auto-regression, ARMA, exponential smoothing, and machine learning approaches [4].

Moving average generally generates poor results for time-series analysis [4]. Therefore, it is usually applied only to remove noise from the time-series. In contrast, auto-regression is largely used in the field. The results in [14] have shown that the performance of auto-regression depends highly on monitoring the interval length, the size of the history window, and the size of the adaptation window. ARMA is a combination of moving average and auto-regression algorithms. The authors in [15] have used ARMA to predict future workload. Machine learning algorithms have been used in [5] and [16]. The authors in [16] have verified NN and Linear Regression algorithms to predict the future value of CPU load and they have concluded that NN surpasses Linear Regression in terms of accuracy. In addition, they have shown that accuracy of both algorithms depends on the input window size. On the other hand, the authors in [5] have evaluated different machine learning prediction results. The authors have considered SVM, NN and Linear Regression. They have verified the prediction results of these algorithms using three performance metrics: CPU utilization, throughput, and response time.

In this paper we used SVM and NN techniques for the prediction task. These methods are two of the most accurate machine learning algorithms in the cloud auto-scaling field [5] and have been widely used in other engineering fields. In section 3 mathematical foundations of these algorithms (i.e., SVM and NN) and their specific configuration in our experiment is presented.

III. Support Vector Machine and Neural Network

Support Vector Machine and Neural Networks are supervised learning models that can be used for classification and regression analysis. The objective of this paper is to assess regression capabilities of these algorithms. Sub-sections III.A and III.B, respectively, present a brief description of SVM and NN as well as their special configuration that was used in our experiment.

A. Support Vector Machine

The purpose of this sub-section is to provide a brief description of the mathematical foundation of SVM. Readers are encouraged to see [17] for more details on SVM. Support Vector Machine (SVM) is used for many machine learning tasks such as pattern recognition, object classification, and regression analysis in the case of time series prediction. Support Vector Regression, or SVR, is the methodology by which a function is estimated using observed data, which in turn "trains" the SVM. In this paper we use SVR and SVM terms interchangeably.

Fundamentally, the goal of time series prediction is to estimate some future value based on current and past data samples. Mathematically stated:

$$x^{\wedge}(t + \Delta t) = f(x(t-a), x(t-b), x(t-c), \ldots) \quad (1)$$

Where $x^{\wedge}$ is the predicted value of a discrete time series x. The goal of time series prediction is to find a function f(x) such that $x^{\wedge}$ is unbiased and consistent.

For a time series prediction algorithm, equation (1) defines a function f (x) that will have an output equal to the predicted value for some prediction horizon. By using regression analysis, equations (2) and (3) define these prediction functions for linear and non-linear regression applications respectively:

$$f(x) = (w \,.\, x) + b \quad (2)$$
$$f(x) = (w \,.\, \varphi(x)) + b \quad (3)$$

If the data is not linear in its "input" space, the goal is to map the data x(t) to a higher dimension "feature" space, via φ(x) (referred to as a Kernel Function), then perform a linear regression in the higher dimensional feature space. Given equations 1, 2 and 3, SVM training goal is to find "optimal" weights *w* and threshold *b* as well as to define the criteria for finding an "optimal" set of weights. First criterion is the "flatness" of the weights, which can be measured by the Euclidean norm (i.e. minimize $||w||^2$) and second criterion is the error generated by the estimation process of the value, also known as the structural risk, which is to be minimized. The overall goal is then the minimization and the regularized risk $R_{Reg}(f)$ (where f is a function of x(t)) defined as:

$$R_{Reg} = R_{structural}(f) + \frac{\lambda}{2}||w||^2 \quad (4)$$

The scale factor λ is commonly called the regularization constant and this term is often referred to as the capacity control term. Its function is to reduce "over-fitting" of data and minimize bad generalization effects. See sub-section III.B for more details on over-fitting.

In our experiment we used SVM algorithm in WEKA tool, which is called SMOreg. See [18] for more details about WEKA project. SMOreg algorithm has four kernels that can be used to train a model: Linear, Polynomial, Radial Basis Function (RBF) and Sigmoid. In our experiment we used RBF kernel, because RBF is the most suitable kernel to handle the case when the relationship between features and target value is nonlinear. In addition, we used default value for the complexity parameter, which equals to 1. The complexity parameter controls the trade-off between errors of the SVM on training data and margin maximization [17].

B. Neural Networks

A neural network (NN) is a two-stage regression or classification model, typically represented by a network diagram [19]. Some of the most commonly used neural network classifiers (algorithm) are: feed-forward, back-propagation, time delay, and error correction neural network

classifier. According to [19], there is typically one output unit Y_1 at the top layer (i.e. K=1) of each neural network for regression problems though multiple quantitative responses can be handled in a seamless fashion. Derived features Z_m are created from linear combinations of the input, and then the target Y_k is modeled as a function of linear combinations of the Z_m,

$$Z_m = \sigma\,(\alpha_{om} + \alpha_m^T X), \mathrm{m} = 1, \dots, \mathrm{M} \quad (5)$$

$$T_k = \beta_{0k} + \beta_k^T Z, k = 1, \dots, K \quad (6)$$

$$f_k(x) = g_k(T), k = 1, \dots, K \quad (7)$$

where $Z = (Z_1, Z_2, \dots, Z_m)$ and $T = (T_1, T_2, \dots, T_m)$.

The activation function σ(v) is usually chosen to be the Sigmoid $\sigma\,(v) = \frac{1}{(1+e^v)}$. Sometimes, Gaussian radial basis functions are used as activation function, producing what is known as a radial basis function network [19]. The units in the middle of the network, computing the derived features Z_m, are called hidden units because the Z_m values are not directly observed. Neural network model has unknown parameters, often called weights, and the goal of training phase is to seek values for weights that make the model fit the training dataset well. Neural networks use empirical risk minimization to find the most suitable weights. Empirical risk is defined as:

$$R_{emp}(f) = \frac{1}{N}\sum_{i=0}^{N-1} L(\,f(x_i), y_i) \quad (8)$$

Where *i* is an index to a discrete time series t = {0, 1, 2,…, N-1} and y_i is the truth data (i.e. training dataset) of the predicted value being sought. L(.) is a lost function or cost function. For more details about cost function see [17].

One of the most significant challenges of using neural networks for regression is over-fitting. Over-fitting occurs when a regression model describes a noise data instead of underlying relationship [19]. Usually neural networks have too many weights, which results in over-fitting the data at the global minimum. Therefore, an early stopping rule is used to avoid over-fitting by training the model only for a while and stopping well before model approaches the global minimum.

In our experiment we used neural network algorithm defined in the WEKA tool, which is called MultiLayerPerceptron. The parameters we used in our experiment are shown in Table I. Parameter selection is usually based on heuristics, as there is no mathematical formula or theory that has been proposed to select the best parameters. We have selected MultiLayerPerceptron parameters based on the best results after several trials.

TABLE I. MULTILAYERPERCEPTRON PARAMETERS USED IN OUR EXPERIMENT

Parameter Name	Value
Learning Rate (ρ)	0.38
Number of hidden layers	1
Number of hidden neurons	4
Momentum	0.2
Epoch (training time)	10000

IV. EXPERIMENT AND RESULTS

The ultimate goal of this experiment is to improve prediction accuracy of predictive auto-scaling systems. SVM and NN are two of the most accurate machine learning algorithms [5] that can be used for workload prediction. Each of these algorithms has its powers and weaknesses, which makes it suitable for specific environments. In this experiment we aimed to explore relations between different workload patterns and prediction accuracy of SVM and NN. Results of this experiment can help us to design and implement a prediction suite which uses the most suitable prediction algorithm based on incoming workload pattern.

A. Experimental Environment

To setup the experiment environment we deployed Java implementation of TPC-W benchmark on Amazon EC2 infrastructure. Fig. 2 illustrates architectural overview of our experimental setup. As shown in Fig. 2, the experimental setup consists of three virtual machines running on Ubuntu Linux for the client (RBE), web server and database, respectively. Table II presents details of these virtual machines. Note that to decrease experiment complexity we have only monitored performance of the web server tier in this paper and assumed that database is not a bottleneck. Hence, a relatively powerful virtual machine is assumed to be dedicated to the database tier.

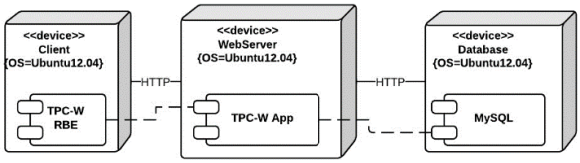

Fig. 2. Architectural overview of experiment

TABLE II. HARDWARE SPECIFICATION OF SERVERS FOR EXPERIMENT

	Memory	Processor	Storage
Client	1 GB	4 core	8 GB
Web server	1 GB	4 core	8 GB
Database	2 GB	8 core	20 GB

On the client side, customized scripts were used along with TPC-W workload generator to produce "growing", "periodic", and "unpredicted" workload patterns. Each of the workload patterns was generated for 300 minutes (i.e. experiment duration = 300 minutes). Then, on the web-server machine, total number of user requests was stored in log files every minute, which resulted in a workload trace file with 300 data points for each of the workload patterns. We refer to these workload trace files as "Actual Workloads" in the rest of this paper.

B. Training and Testing of SVM and NN

After generating actual workloads, SVM and NN were trained and tested. SVM and NN are samples of supervised learning category of machine learning techniques. In supervised learning – unlike unsupervised learning – a prediction (or regression) model is built based on a predefined training dataset. Then, the created model is evaluated against

the testing dataset. According to [5] in auto-scaling area the optimum training duration for SVM and NN is 60% of the experiment duration. Therefore, we considered the first 180 data points (i.e. 60%) of the actual workload trace files as training datasets and the rest 120 data points as testing datasets.

Another important factor in training and testing SVM and NN is data features. In machine learning a feature is an individual measurable property of a phenomenon being observed. Technically, regression algorithms discover relations between features and use these relations to predict future value. For example in an environment with features A and B, a regression algorithm uses training dataset (e.g. training dataset = {(a1, b1), (a2, b2),…, (ai, bi)}) to extract relations between samples of A and B. Then, regression algorithm uses those extracted relations along with the future value of A (e.g. a_{i+1}) to predict future value of B (e.g. b_{i+1}). In our experiment, actual data has only one feature, which is number of requests per minute. Therefore, in order to use machine learning prediction algorithms we had to either add one – or more than one – feature to our dataset or use sliding window technique. The sliding window technique maps an input sequence *x* of size *k* to an individual output *y* by using a prediction model [16]. Based on our previous example using the sliding window technique, prediction algorithm uses the last *k* samples of B (i.e. [b_1, b_2, …,b_k]) to predict b_{k+1}. Similarly, to predict b_{k+2}, the sliding window technique updates by adding the actual value of b_{k+1} and removing the oldest value from the window (i.e., the new sliding window = [b_2, b_3, …,b_{k+1}]). Although sliding window technique has proved successful in the financial and health informatics area [16], setting sliding window size is not a trivial task. Usually smaller window sizes do not reflect correlation between data samples thoroughly, while using greater window size increase chance of over-fitting. Thus, in this experiment – in addition to our main goal (i.e. influence of different workload patterns on SVM and NN prediction accuracy) – we studied effect of sliding window size on the prediction accuracy of SVM and NN, too.

To prevent over-fitting problem, we used cross-validation technique for training phase. Cross-validation is one of the most common error estimation techniques where each observation y_i, $i = 1, 2, \ldots, n$ in the sample dataset of size *n* is successively taken out and the remaining $n-1$ observations of the set are used to train the prediction model to estimate the future data point. Finally, the actual output y_i is used to validate the predicted output $y\hat{}_i$ inferred by the fitted model [16]. Readers are encouraged to see [20] for more details about cross-validation technique.

C. Evaluation Metrics

Accuracy of the results can be evaluated based on different metrics such as Mean Absolute Percentage Error (MAPE), Root Mean Square Error (RMSE), PRED (25) and R2 Prediction Accuracy [21]. Among these metrics, PRED(25) only considers percentage of observations whose prediction accuracy falls within 25% of the actual value. In addition, R2 Prediction Accuracy is a measure of goodness-of-fit, which its value falls within the range [0, 1] and is commonly applied to linear regression models [16]. Due to the limitations of PRED (25) and R2 Prediction Accuracy, we used MAPE and RMSE metrics in this work. Formal definitions of these metrics are:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|YP_i - Y_i|}{Y_i} \quad (9)$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(YP_i - Y_i)^2}{n}} \quad (10)$$

Where YP_i is the predicted output and Y_i is the actual output for i-th observation, and *n* is the number of observations for which the prediction is made. MAPE usually expresses accuracy as a percentage and is a popular metric in statistics, especially in trend estimation. RMSE represents the sample standard deviation of the differences between predicted values and observed values. Smaller MAPE and RMSE values indicate a more effective prediction scheme.

D. Results

Tables III and IV represent the training and testing accuracy of SVM and NN in predicting "periodic" workload pattern.

TABLE III. MAPE AND RMSE VALUES FOR SVM AND NN IN TRAINING PHASE. (PERIODIC PATTERN)

Windows size	SVM MAPE Training	SVM RMSE Training	NN MAPE Training	NN RMSE Training
2	2.7604	5.3701	4.9108	7.044
3	2.7295	5.3646	5.1113	7.3232
4	2.7356	5.3725	4.8585	6.7266
5	2.7391	5.3706	4.8553	6.6891
6	2.7319	5.3706	4.5919	7.0763
7	2.7389	5.3713	4.6787	6.7764
8	2.7339	5.3507	4.3998	6.5587
9	2.6765	5.1351	4.498	7.0429
10	2.6742	5.1534	3.3097	5.6496

TABLE IV. MAPE AND RMSE VALUES FOR SVM AND NN IN TESTING PHASE (PERIODIC PATTERN)

Windows size	SVM MAPE Test	SVM RMSE Test	NN MAPE Test	NN RMSE Test
2	3.7998	6.1277	4.0323	6.2515
3	3.7555	6.11	4.242	6.4143
4	3.7223	6.08	4.8105	6.5811
5	3.7239	6.0805	4.7989	6.5941
6	3.7225	6.0825	4.8853	6.6911
7	3.7178	6.0727	4.915	6.703
8	3.699	6.026	5.105	6.9418
9	3.869	6.2336	6.2338	8.6962
10	3.8164	6.2002	7.1964	10.6854

Figs.3 and 4 compare MAPE and RMSE values for SVM and NN in the testing phase. It should be noted that due to the space limits, we have only presented MAPE and RMSE values of SVM and NN in the testing phase.

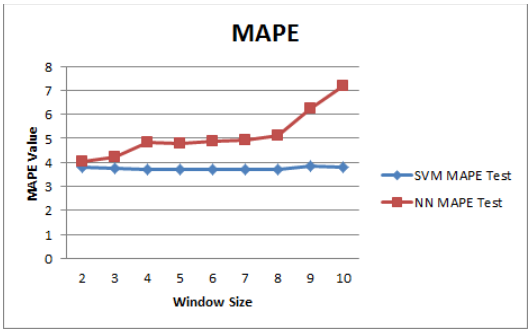

Fig. 3. SVM and NN MAPE values in test phase (periodic pattern)

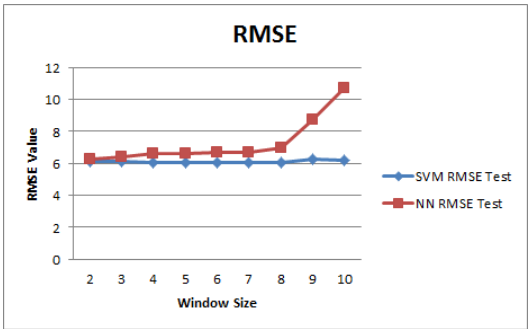

Fig. 4. SVM and NN RMSE values in test phase. (periodic pattern)

According to Tables III and IV SVM prediction accuracy does not change with increasing window size in the testing and training phases. On the other hand, increasing window size has a negative effect on NN prediction accuracy in the testing phase (c.f. Table IV), while its accuracy increases slowly by increasing window size in the training phase (c.f. Table III). The negative effect of increasing window size on NN is due to the "over-fitting" issue. By increasing window size, NN prediction model gets over-fitted to the training dataset. That is why increasing window size increases NN accuracy in the training phase. But, since the prediction model is over-fitted to the training dataset, its prediction accuracy decreases over the testing dataset. Therefore, it can be concluded that for periodic workloads, increasing window size not only does not have positive effect on prediction accuracy but also leads to over-fitting issue which decrease NN accuracy.

In addition, based on Figs. 3 and 4, for small window sizes SVM has slightly better prediction accuracy compared to NN (for window size = 2, SVM MAPE value is 3.80, which is slightly less than NN MAPE value that is 4.03). But, by increasing window size, NN accuracy decreases while SVM accuracy stays steady. Our results show that for environments with "periodic workload pattern" SVM outperforms NN. Moreover, increasing window size does not improve prediction accuracy for this workload pattern.

Prediction results for "growing" workload pattern are shown in Tables V and VI and Figs. 5 and 6. Similar to the results of "periodic pattern", in the environments with growing workload pattern, SVM prediction accuracy does not change with increasing window size. On the other hand, NN prediction accuracy has a decreasing trend but does not change too much with increasing window size. Although SVM and NN have similar reaction to increasing window size, Figs. 5 and 6 show a relatively huge difference in their prediction accuracy. According to Figs. 5 and 6 SVM has much better prediction accuracy compared to NN in environments with "growing" workload pattern.

TABLE V. MAPE AND RMSE VALUES FOR SVM AND NN IN TRAINING PHASE. (GROWING PATTERN)

Windows size	SVM MAPE Training	SVM RMSE Training	NN MAPE Training	NN RMSE Training
2	1.478	2.5427	1.8441	2.6973
3	1.5043	2.5717	1.8964	2.7509
4	1.5193	2.5918	1.9498	2.8445
5	1.5147	2.5891	2.0025	2.9152
6	1.5359	2.599	2.0189	2.9571
7	1.5547	2.6051	2.0175	2.9613
8	1.5505	2.5992	1.9976	2.9123
9	1.5597	2.6142	2.0116	2.9304
10	1.5652	2.625	2.0504	2.9783

TABLE VI. MAPE AND RMSE VALUES FOR SVM AND NN IN TESTING PHASE (GROWING PATTERN)

Windows size	SVM MAPE Test	SVM RMSE Test	NN MAPE Test	NN RMSE Test
2	1.4923	2.3106	2.6389	3.5268
3	1.4947	2.3087	2.81	3.7651
4	1.5019	2.3263	2.8954	3.8768
5	1.4891	2.3121	3.3506	4.446
6	1.4903	2.2958	3.0118	3.9198
7	1.4869	2.2885	3.1808	4.1743
8	1.4973	2.2953	3.7437	5.0224
9	1.5014	2.2984	3.2206	4.2567
10	1.4965	2.2926	3.2159	4.2219

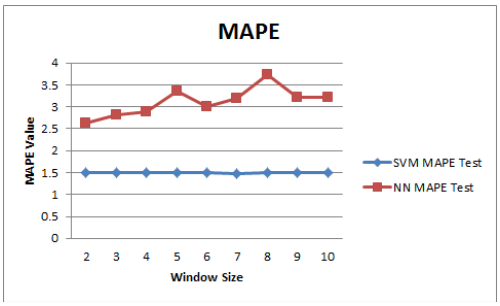

Fig. 5. SVM and NN MAPE values in test phase (growing pattern)

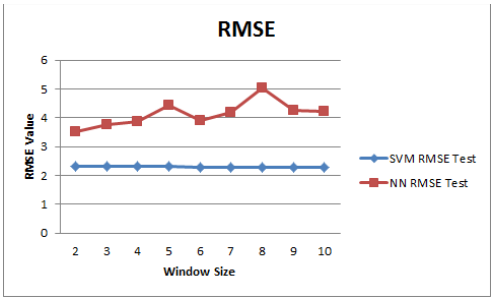

Fig. 6. SVM and NN RMSE values in test phase. (growing pattern)

The last step in this experiment was to analyze SVM and NN prediction accuracy for "unpredicted" workload pattern. By definition, unpredicted workload pattern represents workloads with many consecutive fluctuations [9]. See sub-section II.A for more details. Tables VII and VIII and Figs. 7 and 8 represent our experimental results for unpredicted workload pattern. As shown in Tables VII and VIII, unlike growing and periodic patterns, increasing window size has positive effect on SVM and NN prediction accuracies for unpredicted workload pattern. The reason is that in unpredicted environments there are many fluctuations in the data, therefore, prediction model cannot extract relationships between features thoroughly (in our experiment features are data samples in the window size). Thus, increasing window size increases input size of the algorithm, which leads to improving its prediction accuracy.

Based on Figs. 7 and 8, NN has better prediction accuracy compared to SVM for environments with unpredicted workload pattern. Technically, NN tries to cover all the noise in the data while SVM tries to remove them and find a smooth curve which covers valuable – not noise – data. In environments with too many fluctuations, SVM assumes spikes as being noises and does not capture them. Therefore, in environments with unpredicted workload pattern NN outperforms SVM.

TABLE VII. MAPE AND RMSE VALUES FOR SVM AND NN IN TRAINING PHASE. (UNPREDICTED PATTERN)

Windows size	SVM MAPE Training	SVM RMSE Training	NN MAPE Training	NN RMSE Training
2	0.9207	1.2767	0.8378	1.1917
3	0.8631	1.2364	0.8753	1.2294
4	0.8327	1.2257	0.8811	1.2356
5	0.8089	1.2257	0.8747	1.2322
6	0.7904	1.206	0.9039	1.2527
7	0.776	1.1751	0.9353	1.2625
8	0.7022	1.0563	0.8354	1.1185
9	0.6984	1.0641	0.8424	1.1155
10	0.7017	1.071	0.8485	1.1091

TABLE VIII. MAPE AND RMSE VALUES FOR SVM AND NN IN TESTING PHASE

Windows size	SVM MAPE Test	SVM RMSE Test	NN MAPE Test	NN RMSE Test
2	1.4386	1.7339	1.0216	1.3278
3	1.2357	1.5713	0.9707	1.3217
4	1.1271	1.4782	0.9631	1.3138
5	1.049	1.4114	0.9535	1.3087
6	0.9655	1.3433	0.95	1.3163
7	0.9077	1.2869	0.9488	1.2975
8	0.8	1.1175	0.8335	1.1214
9	0.7908	1.1057	0.7369	1.007
10	0.7999	1.1145	0.719	0.9931

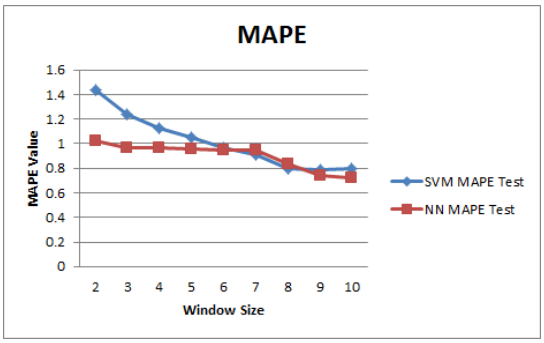

Fig. 7. SVM and NN MAPE values in test phase (Unpredicted pattern)

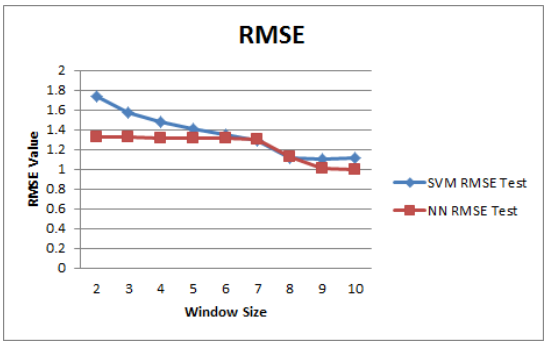

Fig. 8. SVM and NN RMSE values in test phase. (Unpredicted pattern)

According to this experiment's results, it is better to use SVM in the environments with "growing" and "periodic" workload patterns. In addition, increasing window size does not improve SVM accuracy in these environments. On the other hand, for environments with "unpredicted" workload pattern it is better to use NN with greater window sizes. In section V, high level design for a self-adaptive prediction suite is represented which automatically chooses most suitable prediction technique as well as sliding window size based on incoming workload pattern.

V. SELF-ADAPTIVE WORKLOAD PREDICTION SUITE

In this section high level architectural design of a self-adaptive workload prediction suite is presented. The self-adaptive system is based on autonomic system. Based on the experiment results in section IV, the proposed suite chooses automatically NN for workload predictions in environments with unpredicted workload pattern, and SVM for environments with periodic or growing workload patterns.

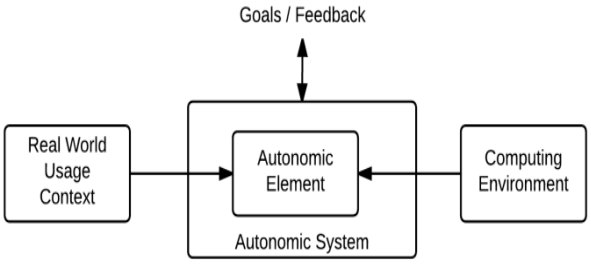

Fig. 9. Classical autonomic system

The goal of a classical autonomic system (shown in Fig. 9) is to make computing systems self-managed and this is motivated by the increasing complexity in software systems due to objects (i.e. run time) change, environmental influence, and ownership cost of the software [23][24]. The objective is to allow human (i.e. software designers and administrator) to specify the system's high-level business goals (or algorithms) and for these to serve as "rules" for the autonomic processes. The term autonomic implies something occurring automatically from internal causes and in the case of a software system, business rules are built into the software. One of the major requirements for operating computer system with minimal human intervention is that a computer system must be able to monitor its activities at runtime based on its internal situation. In addition, it must be able to keep the knowledge about its past, present, and future goals.

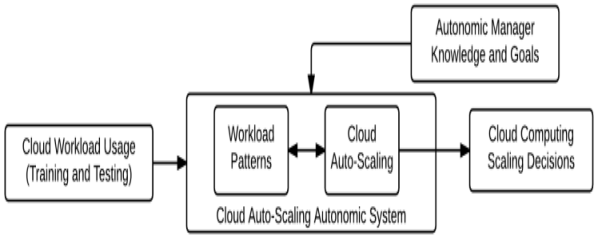

Fig. 10. Cloud workload scaling autonomic system

The idea is that a self-managed system (i.e. autonomic system) must be attentive to its internal operation and adapt to the "behavior" change in order to produce "future" actions.

A typical autonomic system [25], [26], [27] consists of a "context," autonomic element(s), and the computing environment (see Fig. 9). In addition, the autonomic system receives goals and gives feedback to external environment. Autonomic system depends on the autonomic element(s) which is an executable function of the system that exhibits autonomic properties. An autonomic element will regularly sense sources of change by using "sensors" or "reasons" about the current system situation. In our case, the "reason" (or sensor) is the change in workload pattern. (cf. Fig. 10). An autonomic element senses the sources of change and reasons about the current situation for possible adaptation and action for the future. In some cases an autonomic element can be the complete system and in this case the autonomic element and the autonomic system are the same. In general, the situation can be more complex in which case an autonomic system consists of several autonomic elements.

In this work, we have adapted the classical autonomic system architecture for our cloud auto scaling autonomic system architecture. The mapping between the two is shown in Fig. 11.

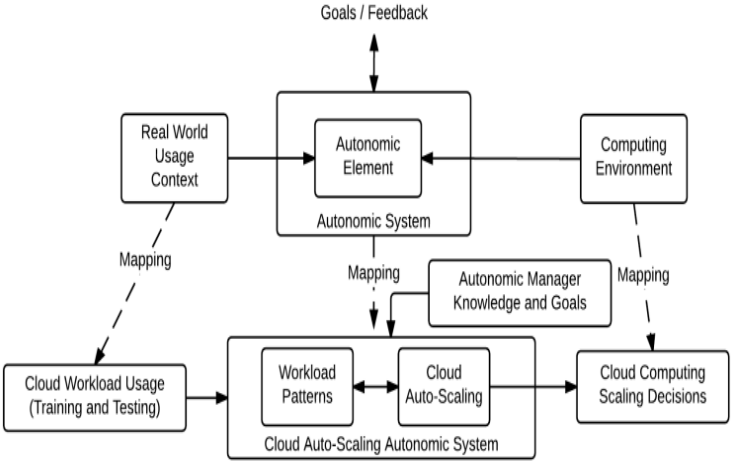

Fig. 11. Mapping classical autonomic system to cloud workload scaling autonomic system

Our cloud auto scaling architecture consists of cloud workload context; cloud auto scaling autonomic system consisting of two meta-autonomic elements (workload pattern and cloud auto scaling); and cloud computing scaling decisions. In addition, we have added a component for autonomic manager, knowledge, and goals.

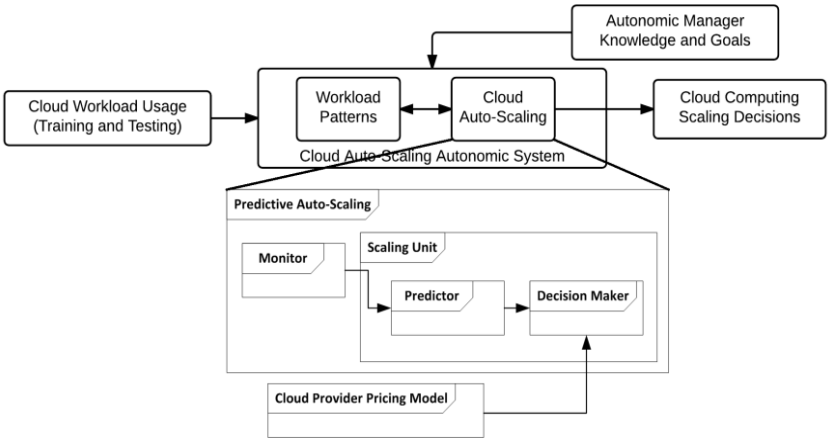

Fig. 12. Projection of the cloud auto-scaling autonomic element

The cloud workload usage represents the "real world usage context" while the cloud computing scaling decisions represents the "computing environment" context. It is important to note that an autonomic system will always operate and execute within a context. The context is in general defined by the environment and the runtime behavior of the system. The purpose of the autonomic manager is to apply the domain specific knowledge linked to the cloud workload pattern and apply the appropriate predictor algorithm (cf. Fig. 12) in order to foresee (i.e. predict) the future. It is constructed around the analyze/decide/act control loop. In addition, Fig. 12 shows the projection of the cloud auto scaling autonomic element.

The cloud auto scaling autonomic elements (workload patterns and predictor a part of the cloud auto scaling) are designed such that the architecture can be implemented using strategy design pattern (cf. Fig. 13). Strategy design pattern consists of a "strategy" and a "context." In our case (cf. Fig. 13) the predictor is the strategy and the workload pattern is the context. In general strategy and context interact to implement the chosen algorithm. A context passes all data (i.e. the workload pattern) required by the algorithm to the strategy. In our case, the context passes itself as an argument to the strategy and this lets the strategy call on context as required. The way this works is that the context determines the workload pattern and passes its pattern interface to the strategy's interface. The strategy then uses the interface to invoke the appropriate algorithm based on the workload pattern interface. All this will be done at runtime automatically and this is what makes the auto scaling system to be an autonomic system. A careful examination of the strategy design pattern (cf. Fig. 13) will show that the context is in turn designed using a template method design pattern. The intent of a template method is to define the skeleton of an algorithm (or function) in an operation deferring some steps to subclasses. In a generic strategy design pattern, the context is simply an abstract class with no concrete subclasses. We have modified this by using template method to introduce concrete subclasses to represent the different workload patterns and to implement the workload pattern context as an autonomic element. This way, the cloud work load pattern is determined automatically and the pattern interface is passed on to the predictor autonomic element which then invokes [automatically] the appropriate prediction algorithm for the workload pattern.

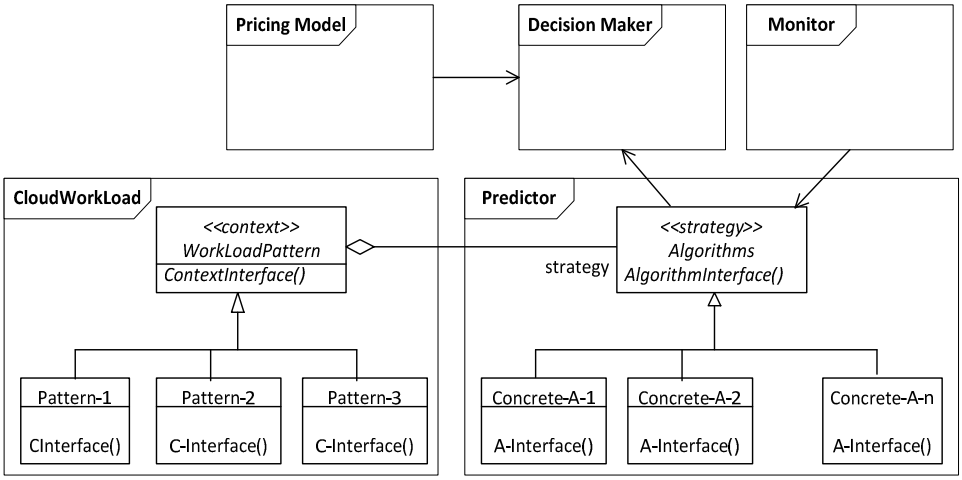

Fig. 13. Design of autonomic elements (Cloud workload pattern and predictor) using strategy design patter.

VI. Conclusions and Future Research

In this paper, we have proposed and proved a hypothesis on prediction accuracy of predictive auto-scaling systems for the IaaS layer of cloud computing. According to the hypothesis, prediction accuracy of predictive auto-scaling systems can be increased by choosing an appropriate time-series prediction algorithm based on incoming workload pattern. In the experiment, the influence of workload patterns on prediction accuracy of SVM and NN was studied by using RMSE and MAPE factors as accuracy assessment criteria. Our findings showed that in the environments with "growing" or "periodic" workload patterns SVM has better prediction accuracy compared to NN, while in the environments with "unpredicted" workload patterns NN outperforms SVM. Our results also showed that increasing the sliding window size only has impact on the environments with "unpredicted" workload pattern with increase in prediction accuracy of SVM and NN. However, in other environments (i.e., growing or periodic workload patterns), increasing the window size does not improve the SVM and NN accuracy.

The experiment results demonstrated that it is ineffective to adopt a system that uses only one particular prediction technique for all environments. Therefore, we proposed a self-adaptive prediction suite based on empirical studies, which facilitates the selection of the most appropriate prediction algorithm as well as sliding window size based on the incoming workload pattern. Furthermore, the novelty of the proposed architecture is that the usage of strategy and template design patterns guarantees automatic runtime selection of the appropriate prediction algorithm as well as detection of workload pattern and window size.

This paper proposed architecture for the self-adapting auto-scaling prediction suite using relevant design patterns. We are currently developing the self-adaptive auto-scaling prediction suite. In addition, in this work we focused on the prediction side of predictive auto-scaling systems, while the exact impact of increasing prediction accuracy on the final scaling decision is unknown. Thus, investigating sensitivity of auto-scaling system vis-à-vis different workload predictions can be considered as another future work. Lastly, in this current work, we assume that the database layer has no negative impact on the prediction by assigning a bigger and better processor and memory. A further future work will be to study the impact of the database layer and latency on the prediction and decision making accuracy of the different algorithms based on workload patterns as well as the different window sizes.

REFERENCES

[1] P. Mell, T. Grance, "The NIST definition of cloud computing," NIST special publication, pp 800-145, 2011.

[2] A. Y. Nikravesh, S. A. Ajila, C.H. Lung, "Cloud resource autoscaling system based on Hidden Markov Model (HMM)", Proc. of the 8th IEEE International Conference on Semantic Computing, June 2014.

[3] A. A. Bankole, "Cloud client prediction models for cloud resource provisioning in a multitier web application environment", Master of Applied Science Thesis, Electrical and Computer Engineering Department, Carleton University, 2013.

[4] T. Lorido-Botran, J. Miguel-Alonso, J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," Journal of Grid Computing, vol. 12, no. 4, December 2014.

[5] A. Y. Nikravesh, S. A. Ajila, C. H. Lung, "Measuring prediction sensitivity of a cloud auto-scaling system", Proc. of the 7th IEEE International Workshop on Service Science and Systems, in collaboration with the 38th International Computers, Software & Applications Conference, July 2014.

[6] S. A. Ajila, A. A. Bankole, "Cloud client prediction models using machine learning techniques," Proc. of the IEEE 37th Computer Software and Application Conference, 2013.

[7] Workload Patterns for Cloud Computing, 2010, [Online], Available: http://watdenkt.veenhof.nu/2010/07/13/workload-patterns-for-cloudcomputing/.

[8] J. R. Williams, F. R. Burton, R. F. Paige, F. C. Polak, "Sensitivity analysis in model-driven engineering," Proc. Of the 15th International Conference on Model Driven Engineering Languages and Systems, 2012.

[9] H. W. Cain, R. Rajwar, M. Marden, M. H. Lipasti, "An architectural evaluation of Java TPC-W," Proc. Of the 7th International Symposium on High Performance Computer Architecture, 2001.

[10] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter, C*loud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*, Springer, 2014.

[11] Amazon Elastic Compute Cloud (Amazon EC2), 2013. [Online], Available: http://aws.amazon.com/ec2.

[12] RackSpace, The Open Cloud Company, 2012. [Online], Available: http://rackspace.com.

[13] RightScale Cloud management, 2012. [Online], Available: http://rightscale.com.

[14] M. Z. Hasan, E. Magana, A. Clemm, L. Tucker, S. L. D. Gudreddi, "Integrated and autonomic cloud resource scaling," Proc. Of the Network Operations and Management Symposium, 2012.

[15] J. Kupferman, J. Silverman, P. Jara, J. Browne, "Scaling into the cloud," Technical Report, Computer Science Department, University of California, Santa Barbara, 2009.

[16] N. Roy, A. Dubey, A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," Proc. Of the 4th IEEE International Conference on Cloud Computing, 2011.

[17] S. Islam, J. Keung, K. Lee, A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," Future Generation Computer Systems, vol. 28, no. 1, pp 155 – 165, 2012.

[18] Nicholas I. Sapankevych, R. Sankar, "Time Series Prediction Using Support Vector Machines: A Survey", IEEE Computational Intelligence Magazine, vol. 4, no. 2, May 2009.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, H. Witten, "The WEKA data mining software: an update," SIGKDD Explorations, vol. 11, no. 1, 2009.

[20] Trevor, H., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, New York: Springer, February, 2009.

[21] S. Arlot, A. Celisse, "A survey of cross-validation procedures for model selection", Journal of Statistics Surveys, vol. 4, pp 40–79, 2010

[22] I. Witten, E. Frank, "Data mining practical machine learning tools and techniques with Java implementations," Academic Press, San Diego, 2000.

[23] D. Garlan, B. Schmerl, "Model-based adaptation for self-healing systems", Proc. of the 1st Workshop on Self-Healing Systems, 2002.

[24] R. Sterritt, B. Smyth, M. Bradley, "PACT: personal autonomic computing tools", Proc. of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pp. 519–527, 2005

[25] J.P. Bigus, D.A. Schlosnagle, J.R. Pilgrim, W.N. Mills III, Y. Diao, "ABLE: a toolkit for building multiagent autonomic systems". IBM Syst. Journal, vol. 41, no. 3, pp 350–371, 2002

[26] M.L. Littman, N. Ravi, E. Fenson, R. Howard, "Reinforcement learning for autonomic network repair", Proc. of the 1st International Conference on Autonomic Computing, pp. 284–285, 2004.

[27] J. Dowling, E. Curran, R. Cunningham, V. Cahill, "Building autonomic systems using collaborative reinforcement learning", Knowledge Eng. Rev. no. 21, pp 231–238, 2006.

[28] K. Hwang, X. Bai, Y. Shi, M. Li, W. Chen, Y. Wu, "Cloud Performance Modeling and Benchmark Evaluation of Elastic Scaling Strategies," IEEE Transactions on Parallel and Distributed Systems, vol. PP, no. 99, 2015