ACAS: An Anomaly-based Cause Aware Auto-Scaling Framework for Clouds

Sara Kardani Moghaddam, Rajkumar Buyya and Kotagiri Ramamohanarao
Cloud Computing and Distributed Systems (CLOUDS) Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia

Abstract

Cloud computing as a model to deliver distributed resource and services on the pay-as-you-go policy has become increasingly popular for both academia and industry. However, the inherent dynamicity in this environment makes it prone to various types of performance problems which introduce many challenges in the area of distributed resource management. Advances in the big data learning approaches can bring the opportunity for a data aware dynamic management of resources in the cloud. The collected data from the performance indicators of the system can be a valuable source of information to identify unusual behaviours in the resource consumptions or application performance. Different types of problems can cause the performance degradations at VM or system level. System administrators are overwhelmed with the huge amount of data to be processed to find about these problems and overall functionality of the system. In this paper, we argue that a better selection of dynamic resource scaling policies can be achieved for better performance by predicting the anomalies in the system and narrowing down the possible cause of the anomaly to one of the attributes of the system. Therefore, we propose a 2-level cause aware auto-scaling framework which leverages two types of resource management solutions, horizontal and vertical, as the corrective actions when a performance anomaly is detected. We show the effectiveness of vertical scaling strategy as a quick solution for cases that a VM is exposed to some type of the local anomaly, while the horizontal scaling solutions can be used for system wide load problems to add new VMs in the system. Moreover, our data analysis module can predict anomalies to give the scaling system enough time to make an effective scaling decision. The proposed unsupervised anomaly detection module leverages a new updating strategy for renewing the models which considers the changes in the state of the system to reduce the overhead of recurrent model trainings. We have performed a comparison of the proposed framework with an approach which is used by several famous cloud providers to show the advantage of mixing the multi-level auto-scaling with the knowledge of anomaly detection analysis in resolving performance problems in the cloud.

Keywords: Cloud computing, Anomaly Detection, Performance Management, Isolation-Trees, Vertical Scaling

1. Introduction

Cloud computing as a model to deliver computing infrastructure and services is bringing both opportunities and challenges in the area of distributed resources management. A wide range of customers from individuals to commercial companies are leveraging the power of offered services in an on demand pay-as-you-go model. On the other hand, public providers such as Amazon and Google aim at satisfying the demand of their customers as well as meeting the service level agreements (SLA) knowing that the violation of SLAs can cost them money and their reputation.

Elastic resource management offers the providers the ability to dynamically adjust the resources based on the type and the number of requests from users. For example, existing auto-scalers such as Amazon elastic auto-scaler [1] enable the system to dynamically add and remove Virtual Machine (VM) instances as a response to the observed performance degradations in the system. Workload fluctuations targeting hosted applications are one of the main underlying reasons for these performance problems. This is a highly important observation, especially for the large scale web application systems where the interaction between users and web servers can change frequently, affecting the pattern of workloads and resource requirements. In the other hand, there is a variety of problems that can happen locally in one VM such as a bug in the application code, resource bottlenecks or hardware faults. This type of problems can affect the local performance of the VM adversely. Distinguishing these faults from system wide problems can help auto-scalers to make more informed decisions by focusing on the solutions that target directly the root cause of the anomalies. To achieve this goal, we can divide the data-aware resource management problem into two main subproblems that should be dealt with separately.

First, it should be mentioned that different types of performance problems in the VMs usually leave distinctive signs in the performance indicators of the machine. Therefore, continuously monitoring the behaviour of resources by collecting the values of important attributes provides system administrators a valuable source of the data that can be analyzed to have a timely information on the performance state of the system. Big data learning approaches offer the researchers the necessary concepts and tools to dig into the collected data and find interesting patterns of unexpected behaviours or anomalies with their possible causes.

Second part of the problem focuses on the auto-scaling solutions to be triggered when a performance problem is identified during analyses of the collected data from the system. There is a variety of resource management solutions including horizontal scaling, elastic VM management, migrations, resource contention management, etc. However, when the scaling actions should be triggered and which type of the policy can be selected are different challenges which are investigated in this paper.

With regard to the aforementioned challenges, we propose an Anomaly and Cause Aware auto-Scaling (ACAS) framework consisting of three main modules, monitoring, data analyzer, and resource auto-scaler which leverages two types of resource adjustment policies, horizontal and vertical

scaling. The focus of this work in on the local anomalies such as CPU and memory bottlenecks as well as system wide load problems that can affect the performance of the applications. Accordingly, the **contributions** of this paper are as follows:

- Proposed a proactive, unsupervised anomaly detection on predicted performance data of the VMs to identify future anomalies of the system. We have also developed a strategy for deciding when the models need to be updated to reduce the recurrent model training overheads
- Proposed a two-level auto-scaler by categorizing anomaly problems into two levels of local and global anomalies. Local anomalies are targeted by elastic VM solutions while system-wide anomalies are resolved by global scaler
- Achieved more scalability by breaking down the problem of performance management to local and global anomaly detection
- Extending CloudSim simulator by adding characteristics of the elastic VMs
- Performed extensive experimental evaluation of the proposed system under various loads and demonstrated ACAS ability to maintain a better quality of performance compared to the existing resource management solutions

An extensive set of experiments are done targeting both types of local anomalies and global load problems. The experiments show that distinguishing between VM specific anomalies and system wide load problems help auto-scaler to take advantage of fast vertical scaling policies to increase bottleneck resources for one VM while the proactive anomaly detection helps to trigger early system wide horizontal scaling actions to reduce the number of SLA violations.

The rest of this paper is organized as follows: Section 2 introduces some of the existing works in the field of data-aware resource management. Section 3 presents the motivation, an overview of the approach and a brief introduction to Isolation-Trees based anomaly detection. Section 4 presents the details of learning algorithms and explains communication among the modules. Section 5 presents the experiments and the results and finally, the paper is concluded with the future works in Section 6.

2. Related work

The idea of utilizing data learning techniques for the performance analysis in the cloud has been of great interest to the researchers in recent years. The work presented in [11] investigates the feasibility of Isolation-Trees based anomaly analysis to detect anomalies in data from IaaS data centers, focusing on the behaviour of the algorithm to the presence of seasonality/trends in their dataset. ACAS also leverages the same concept of Isolation-Trees in the anomaly detection part of

the problem. However, ACAS is a complete framework that covers the problems of online learning and model updating, root cause analysis and resource management modules. [12] proposes a method for long-term load prediction in Google data centers, considering load as the main factor involved in resource management solutions. Another work presented by [13] considers a single attribute, number of required processors at a certain time, for resource utilization estimation. [14] presents a regression based workload prediction framework to improve the utilization of the resources while reducing the cost. To achieve this goal, they use the knowledge from workload prediction to decide on the time and amount of resource to be changed in the system, considering both types of vertical and horizontal scaling. [15] combines workload prediction and reinforcement learning to find the best configuration for VM resources. The feedbacks from application performance and resource utilizations are used to calculate the reward and update the resource configuration strategy for better selection of future actions. Compared to our framework, the aforementioned works address the problem of resource management by focusing on the workloads as the only influential factor for performance analysis and ignore other sources of performance problems in the system. [16] follows a more systematic approach to the problem of VM management in the cloud by modeling the problem as a feedback-based control approach. The Proportional-Integral-Derivative (PID) based controller is designed to manage the number of VMs in the system, aiming at keeping the service quality in accordance with the agreement levels. [17] designs a reinforcement learning approach to gradually learn from the environment and decide on the VM level scaling of the system to alleviate the performance problems occurring due to the load fluctuations in the system. Different to our model, these works consider the management of resources only at VM level by changing the number of VMs in the system. [5] presents an automatic anomaly identification technique for adaptively detecting performance anomalies such as Disk and Memory related failures. Proposed method investigates the idea that a subset of the principal components of metrics can be highly correlated to specific failures in the system. BARCA, proposed by [18], is another framework for online identification of anomalies in distributed applications which divides anomaly detection process into two steps. First, a one class classifier is employed to distinguish normal behaviour from unexpected ones. Second, a multi-class classifier is used to separate different types of anomalies from detected abnormal behaviours. [19] investigates proactive anomaly detection in data stream processing systems. The proposed solution includes a phase of predicting resource utilization and then applying an anomaly identification algorithm on the predicted data. The target anomalies are injected and the training is done on a labeled dataset of different anomaly occurrences in the past data. Although these works focus on the same problem of anomaly detection, but how this information can be used for resource management is not investigated. Alternatively, [20] addresses the performance problem by integrating a 2-dependent Markov model as the predictor and tree-augmented Bayesian networks (TAN) for anomaly detection. Based on the knowledge from learning algorithm, they apply some

Table 1: Related works on cloud performance management

Work	Data Analysis Method	Resource Management	Proactive	Unsupervised[1]	Vertical Scaling
[11]	IForest (**AD**)[2]	X	X	✓	-
[12]	Bayes Model (Workload Analysis)	✓	✓	-	X
[16]	Control Theory	✓	X	-	X
[17]	Reinforcement Learning	✓	✓	-	X
[18]	SVM (**AD**)	X	✓	✓	-
[19]	Markov models, Bayes Classifier (**AD**)	X	✓	X	-
[20]	Markov models, Bayes Classifier (**AD**)	✓	✓	X	✓
[21]	Threshold-Based Rules	✓	X	-	✓
[22]	Threshold-Based Rules	✓	X	-	✓
[15]	Reinforcement Learning	✓	✓	-	✓
[23]	Self-Organizing Maps (**AD**)	X	✓	✓	-
Proposed work	Isolation-based Trees (**AD**)	✓	✓	✓	✓

[1] This column is applicable for the works with the focus on anomaly detection

[2] Anomaly Detection

type of the vertical scaling or migration to minimize the performance degradation. [21] focuses on the cost effectiveness of vertical scaling approaches and proposes a threshold based scaling strategy to combine different scaling approaches including self-healing, fine-grained resource scaling and VM level scaling to meet QoS while reducing cloud providers' costs. [22] addresses the problem of shared memory management among multiple VM with the over-subscription approach and elastic VM technique. A threshold based strategy on the value of memory related metrics is utilized to trigger the memory adjustment actions while live migration is used to avoid the SLA violations when the total memory demands of the VMs exceed the available memory of the physical machine. In contrast, ACAS focus on the effectiveness of horizontal and vertical scaling policies by leveraging the capabilities of unsupervised learning approaches for situations that system is exposed to the local and load related anomalies. Another study by [23] investigates unsupervised behaviour learning problem for proactive anomaly detection. The proposed framework leverages Self-Organizing Maps (SOM) to map a high dimensional input space (performance metrics) to a lower dimensional map without losing the structural information of original instances. In contrast, we show that resource management process can make use of the knowledge from proactive anomaly detection and root cause identification to address the specific anomalies occurring in one VM.

Table 1 compares the above mentioned works by highlighting the main components and characteristics of the proposed solutions for the problem of performance analysis and management in large distributed systems.

3. Preliminary

In this section, we explain the motivation and an overview of our approach. Then, a short introduction to the concept of isolation-based anomaly detection is presented.

3.1. Motivation and Approach Overview

Virtualization technologies are the core concept in the functionality of cloud models. The possibility of running many VMs and applications on one physical host brings new opportunities as well as more complexity to the design of these environments.

Resource management concept is much more challenging due to the inherent dynamic nature of cloud environment, where we can host different range of applications with a variety of demands and workload types. This is especially important for the large scale web applications, in which the pattern of the incoming requests from the users can change quickly creating a dynamic environment where the configurations of resources should frequently be adjusted to satisfy the demanded SLAs. Elastic resource management, as a solution for this problem, leverages the VM based scaling of the system known as horizontal scaling. Public cloud providers offer customized policies of horizontal scaling to satisfy the resource demands of the applications based on the load in the system. Even though the VM based scaling policy is a common approach to manage performance problems in the cloud environment, it may not be a proper solution for a different category of performance problems caused by the faults in one VM. For example, consider a situation where a memory-intensive process is started in the same VM hosting a web based application which is consuming all the available memory, ignoring the demands of the web application. Therefore, the lack of the free memory can cause performance degeneration such as longer than usual response times from the web server. The conventional scaling approaches add new VMs into the system even though the problem is not caused by the load growth from a higher number of user requests. Having the same load in the system, newly added instances incur extra costs including both resource and license costs as well as higher resource wastage due to added resources which are not utilized. Considering this scenario, there are other types of the problem that can create similar effects on the utilization of the resources. For example, it is shown that the web applications are prone to many of the performance problems which involves CPU and memory resources [2].

In the other hand, existing auto-scaling solutions such as Amazon elastic auto-scaler are designed in a way that are more suitable to track the changes at the system level. They do not consider VM based problems particularly when the changes in one VM does not have an immediate impact on the average performance of the system. One solution to address this category of problems is to have a resource management solution at fine-grained levels of control. Elastic VM architecture enables on-the-fly tuning of VM resources without turning off the VM which avoids the delays of rebooting the system. Given the above explanations, we formulate the problem as the selection of proper

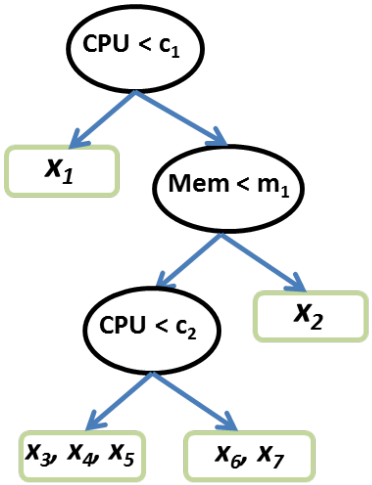


Figure 1: A simple Isolation Tree for two attributes CPU and Memory.

resource scaling policy to satisfy the quality of service (QoS) by analyzing the state of the system to distinguish resource level bottlenecks from system wide load problems. To be more clear about this explanation, we pursue the following definition in the rest of the paper:

System State: State or behaviour of the system at each time is an abstract representation of operational attributes and performance indicators of the system which can be recognized in normal or abnormal/anomalous condition. The main indicators of an abnormal state are the presence of unexpected patterns or values in the load and resource level measurements of VMs and applications.

The proposed framework addresses this problem at the service provider level who has access to the VMs hosting the application to monitor system and application level metrics. In this paper, we target a category of performance anomalies known as resource bottlenecks and particularly two problems insufficient CPU and memory in one VM. Therefore, by tracking resource level metrics of VMs, one can utilize vertical scaling functionality to increase the amount of RAM capacity or the number of CPU cores of one VM to quickly respond to the performance degradations of the system.

In the next section, the components of ACAS framework for cause aware auto-scaling in the cloud are explained in more details. However, before that, we present a brief introduction to IForest algorithm which is the key function of proposed anomaly identification approach in the data analyzer module. It should be highlighted that several other algorithms can also be used for the problem of anomaly detection [3, 4, 5]. However, as we explain more in the following section, IForest shows interesting characteristics as a fast, simple algorithm which makes it a notable alternative to be utilized for real-time tracking of anomalous behaviour.

3.2. A Brief Summary of IsolationTrees

The concept of isolation based anomaly detection which is used in Isolation-Forest (IForest) algorithm [6] is based on the idea that an anomaly instance is isolated in attribute space faster than a normal instance. IForest leverages this definition to create an ensemble of random binary trees on the attribute space of the problem with the expectation that an anomaly instance is isolated quickly close to the root node of the tree.

In order to generate a random tree, IForest first randomly selects a sample of ψ instances and starts by selecting a random attribute with a random value based on the measurements in the selected sample. This step corresponds to creating one node in the tree. Then, sample instances are divided into two categories based on their values for the corresponding attribute. Each category is assigned to a new child node and this process continues for newly created nodes until there is only one instance left or all the instances have the same values or the length of the tree has reached the predefined maximum value. Figure 1 shows a simple Isolation Tree generated on a sample space of 7 instances, each representing the utilization values of CPU and memory of one VM. Let $X = (x_1, x_2, ..., x_7)$ be the set of the input observations for the IForest algorithm. To create the root node, the value of c_1 is selected based on the observed utilization values of CPU attribute in X. As Figure 1 shows, the instance x_1 is the only instance with a utilization less than c_1 which creates a leaf node on the left side of the root node, while the remaining instances create the right child node. The right child node includes more than one instance with different utilization values. Therefore, another attribute is selected to divide the remaining instances, creating two new sub nodes. This process continues until the terminating conditions are met. In this example, instance x_1 can be a possible anomaly as it seems to be in a different range of CPU utilization values compared to the other instances.

In order to build a model, IForest generates t Isolation trees from the training data. An anomaly score can be calculated for each new observation which represents the degree of anomalousness of the record compared to the past observations used for the training of the models. Each instance traverses all the randomly generated trees, starting from the root of the tree until it reaches a leaf node. Then, the score is calculated by averaging the length of the traversed paths on all the tress and using a formula presented in [6]. The anomalous score is inversely proportional to the average length of the branches. Shorter the length, more likely the instance is anomalous. The authors demonstrate that the algorithm has a *low linear-time complexity* with *small memory requirements*. Indeed, as they have shown in the work [6], the worst time and space complexity for training this algorithm is $O(t\psi^2)$ and $O(t\psi)$ respectively, where t is the number of trees in the ensemble which also leads to this conclusion that the training complexity is ***constant*** when the sample size and the number of trees are fixed. The extensive comparisons of IForest done in [6] with other state-of-the-art anomaly detectors such as Random Forests or Local Outlier Factor (LOF) demonstrate the superiority and

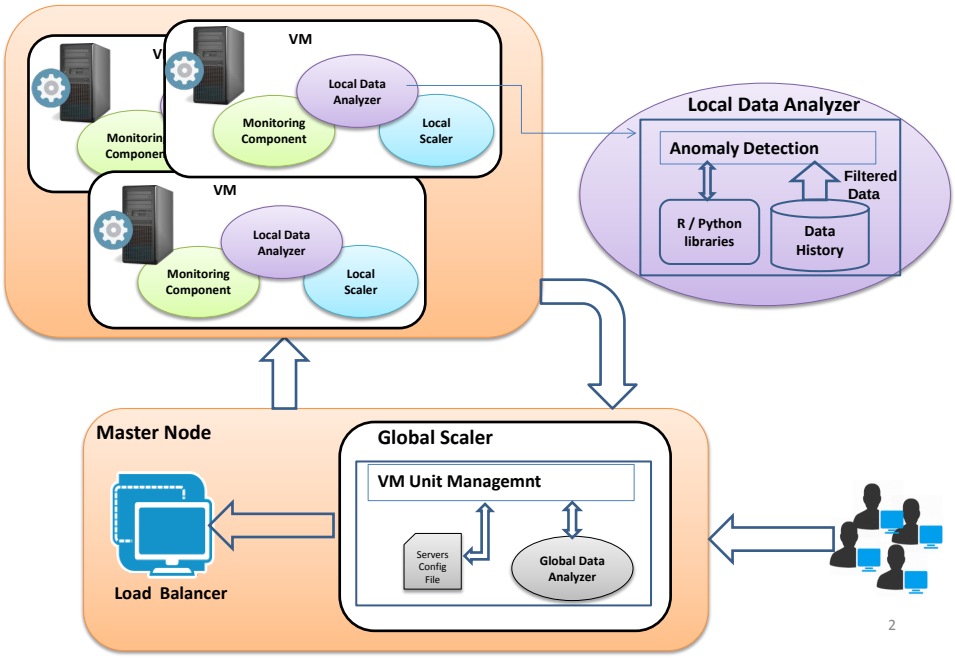


Figure 2: A High Level System Model

robustness of IForest in terms of the accuracy as well as the processing time. Moreover, the authors show that the IForest performs near optimal with the default parameter settings which is consistent with our observations during ACAS experiments. This is a very important feature, especially for highly dynamic environments, where the definition of the models should be updated frequently as the state of the system changes and having an algorithm which does not need time consuming parameter tuning is preferable. Considering the aforementioned characteristics, IForest seems an appropriate choice to satisfy the requirements for having a fast, low overhead anomaly detection module in our framework.

4. System Design

Figure 2 depicts an overview of the proposed framework and how the components work together. The framework is modeled based on a web based application with the application and database servers hosted on the cloud VMs. These applications are shown to have been exposed to many of the performance degradations caused by the changes in the workload or CPU and memory related faults. However, the definitions are generic and can be applied to any distributed application. The components of the application can be distributed on different VMs, while each VM has its own monitoring component, data analyzer and local scaler modules installed. The data analyzer box in the Figure 2 shows the details of the local analyzer module on each VM. The scaling decisions

are performed at two levels, local and global. The local scaler is responsible for the vertical scaling decisions at one VM, while the global scaler performs horizontal scaling decision in the system. The global scaler and the load balancer are parts of a separate master node which plays as the central broker for the whole system. Therefore, the incoming requests are distributed among existing VMs (application servers) based on the load balancer configuration and registered servers at the master.

Each VM monitors the performance of its own resources and collects a variety of attributes such as CPU and memory utilization, and disk I/O rates which can model the state of the system. During regular intervals, collected data are sent to the local data analyzer to be processed for the possible signs of performance problems occurring in the near future. Therefore, at the first step, future values of each metric are predicted. There is a wide range of algorithms which can be used for the prediction and modeling of time series data. We have tested two algorithms ARIMA and feed-forward Neural Networks (NN) for this step and finally selected NN due to the observed stability of its predictions in the presence of the noise in our dataset. NN is utilized to generate a separate model for each metric and predict the future values based on the learned models from the past observations of the system. Upon receiving the newly predicted values, the anomaly detection algorithm calculates an anomaly score for each observation and sends a new alert if the new score exceeds the threshold θ. Before proceeding, we should remark that anomaly detection module considers every deviation in the values of the attributes from the past state of the system as an anomaly which is reflected in the calculated anomaly scores. However, from the service providers perspective, a performance anomaly is important when it shows a possible breach of the SLA objectives; otherwise, it can be ignored. Therefore, to be clear about an anomaly event which is considered by the resource management module for taking the corrective actions we pursue the following definition in the next sections:

Anomaly Event: A continuous change in the behaviour of the system which is reflected as unexpected trends in the values of the monitored attributes of the VM while at least one of the metrics shows the possibility of breaching the threshold for the maximum accepted utilization.

The first part of the aforementioned definition is handled by anomaly detection module to detect the attributes that show a transition in their state based on the details provided in the Section 4.1. The second part of the definition confines the performance anomalies to the anomaly events that are breaching the performance thresholds. This part is considered by resource management module as described in the Section 4.2.

During anomaly detection phase and at the time of observing anomalous behaviour, the system asks the cause detection module to analyze the state of different observed metrics and find a possible cause for detected anomalies. The suggested causes of the problem from this module are used as additional knowledge in the auto-scaler components to help them make more informed decisions regarding the scaling policies.

The results from anomaly detection module are sent to the local and global auto-scaler compo-

nents. The local scaler is responsible for resource configurations at VM level also known as vertical scaling policies. In contrast, the global scaler is aware of the state of the whole system and is responsible for changing the number of VMs in the system known as horizontal scaling policies.Algorithm 1 shows a summary of the main steps of ACAS framework at the local and global level. The details of these steps and the priority of different scaling policies are explained in the following algorithms and subsection. A list of all the notations used in following sections are listed in Table 2.

Algorithm 1: Cause Aware Resource Scaling in ACAS

input : $V = (VM_1, VM_2, , ..., VM_M)$: A list of all registered VMs in the system

1 **while** *The system is running and in the beginning of performance-check interval* **do**
2 **for** $VM_i \in V$ **do**
  `/* This part of the code is executed locally in each VM */`
3 **if** VM_i *has not initialised the IForest models and there are enough data collected for training* **then**
4 Initialize IForest models for VM_i ;
5 **end**
6 Collect the recent monitored values for different metrics of VM_i
7 Call Algorithm 2 on the collected observations to predict future data and find about the possible performance anomalies and suggested causes;
8 call Algorithm 3 to check if VM_i requires a new vertical scaling to be done by local scaler; If scaling is done VM_i goes into a locked state for a predefined time.
9 **end**
  `/* This part of the code is executed in the master node */`
10 Initialize all indicators in f_i to 0;
11 **for** $VM_i \in V$ **do**
12 **if** VM_i *is not in a locked state and is moving to critical condition based on the Algorithm 4* **then**
13 $f_i \leftarrow 1$
14 **end**
15 **end**
16 Decide on a new horizontal scaling action based on the information provided by f_i;
17 **end**

4.1. Anomaly Prediction based on Isolation-Trees models

Given one VM measurements, the goal is to find if the collected values show a different pattern compared to the past behaviour (lines 3-7 Algorithm 1). Therefore, having a sequence of past

Table 2: Description for Notations

Notation	Description
K	Minimum number of alerts before an anomaly record is created
w	prediction window size
lw	Number of observations in prediction learning window
tw	Number of observations in training window
L	Minimum number of violations before the threshold based approach (baseline) starts an action
r	Number of attributes for each observation
LI	Log Time for monitoring system to record a new observation
θ	Threshold for anomaly score. Values greater than θ will be considered as anomaly
th_i	Usage threshold for attribute i
ψ	Number of randomly selected samples from input instances as the input of IForest algorithm

observations from one VM, an ensemble of Isolation-Trees is generated using the IForest algorithm. After the training is done and each VM has the initial models of its performance, the anomaly detection process starts to analyze the new measurements collected from the VM. Algorithm 2 shows the sequence of required steps for the process of anomaly prediction in ACAS. This process is called regularly to check the recent performance of the VM.

In order to give the system enough time to trigger auto-scaling actions, we need to detect anomalies in the future data. Therefore, the first step is to predict the future values of each metric for the VM (lines 1-2). NN algorithm is exploited as the prediction function (f_p) to forecast the w values of each metric based on the recent measurements from the system. Predicted values are fed as the inputs to the trained models which calculate an anomaly score for each predicted record. The anomaly scores show the degree of abnormality of the observations compared to the data used in training phase (line 3).

It should be highlighted here that we expect to encounter cases where ACAS may miss some of the anomalies due to the wrong measurements or wrong predictions resulting from the dynamic nature of the target environment. Therefore, ACAS also considers more reactive mechanisms which try to adjust the scores of anomaly points when a violation in the system is detected. To make this point clear, let S_i be the score for the prediction P_i. ACAS checks if S_i actually reflects the violation observed at time t_i and if it does not (meaning that $S_i < \theta$ and $P_i \geq th_i$), it deliberately

increases the score S_i to a higher value so other components of the framework handle situation as a new anomaly state.

4.1.1. Model Updating

One question to be answered is how the system decides to update the anomaly detector models. The inherent dynamicity in cloud workloads and the possibility of different types of failures highlight the importance of updating models so they can show the most recent state of the system. In this regard, three different states of the system are distinguished as follows:

- **Transition State**: The system is recognized as in transition if it meets two main conditions. First, newly observed values differ from the past training data in the patterns and/or values. Therefore, we expect to see higher anomaly scores calculated to show the abnormality of recent behaviour compared to the historical records. Second, the system has not reached a stable state, meaning that a continuous change of the variables is still observable. The focus of this paper is on the transitions which cause the average values of the attributes to change with the assumption that the patterns remain unaffected. For example, consider a situation that an incremental trend is continuously impacting the values of one of the attributes in the system.

- **Changed State**: The system has reached the changed state when the new observations show deviations compared to the recorded data used for the training. However, the system has reached a stable condition meaning that no significant changes in the average values of the attributes are detected. In terms of the conditions mentioned for the transitions state, a system at changed state satisfies the first condition only.

- **Normal State**: The system is at the normal state when none of the above conditions are satisfied, meaning that the average values of the attributes for recent observations do not show significant changes compared to the training data. As a result, the calculated anomaly scores do not indicate any abnormal behaviour demonstrating a stable environment.

The anomaly detection module decides to update the model if it finds the corresponding VM is at the *changed* state (lines 6-12). The reason is that, at this state, the high number of anomaly alerts shows the previously trained models are not representing the current state of the system. Moreover, the system has reached a new stable environment and new models are required to enable the anomaly detection module to perform in accordance with the changes. The updating procedure continues until the new models correctly reflect the new state or another transition in the system starts. It should be mentioned that ACAS does not consider *transition* state a proper time for updating the models as some of the attributes are showing significant changes in their values and new models quickly become obsolete, resulting in many unnecessary updates.

Algorithm 2: Anomaly Detection

input : $D = (X_1^m, X_2^m, ..., X_{lw}^m), X_i^m \in R^{1\times r}$: A matrix of lw records, each record including measurements for r features

Parameter: w: Prediction Window

θ: Anomaly Score Threshold

output : (Anomaly Alert, Cause of Anomaly)

$c \leftarrow -1$

`/* Prediction function` f_p `is used to predict future values of data.`X^m `corresponds to the Measured data and` X^p `presents Predicted data. */`

$(X_{lw+1}^p, X_{lw+2}^p, ..., X_{lw+w}^p) = f_p(X_1^m, X_2^m, ..., X_{lw}^m)$

$S_i = AnomalyScore(X_{lw+i}^p)$, $0 < i \leq w$: Find the anomaly scores with IForest algorithm. Then, check if these scores should be adjusted based on a reactive approach if some violation is already happening in the system.

$anomalyDetected \leftarrow (Count(S > \theta) > Length(S)/2)$

if *anomalyDetected* **then**

    Initialize all indicators in f_i to 0

    for *feature* $i \in D$ **do**

        if *system is in changing state on dimension* i **then**

            $f_i \leftarrow 1$

        end

    end

    Decide about updating the models based on the information provided by f_i.

    Identify the cause of abnormality and assign it to c.

end

return ($anomalyDetected$, c)

4.1.2. Cause Identification

The cause detection procedure tries to provide some knowledge about the possible resource level root causes of the performance problem to help the scaling modules make a more informed decision about the proper scaling policies. Therefore, if the output scores from anomaly detection module show a possible anomaly is occurring in the VM, the next step is to identify the underlying reason for the problem. The category of changes addressed in this paper are the ones that impact the average values of the attributes with an increasing or decreasing trend. Therefore, to find an attribute with a trend in the values, we follow an approach which fits a regression line on the data and calculates the slope of the line as a measure of the existing trends in the data.

One point worth noting here is how to distinguish load problems from other local anomalies. One observation to be followed is that when the performance of the system is impacted due to the changes in the incoming workload, we expect to see more than one attribute affected and changes their state. Accordingly, ACAS checks whether most of the attributes in the system are recognized at the *transition* state simultaneously and then flags the anomaly as a load problem.

4.2. Resource Management Module

The management of resources in a continuously changing environment requires the integration of resource configuration policies at different layers of granularity. Depending on the type of the problem and identified root causes, some policies may work better at meeting the SLA objectives such as time or cost of the solution. In this paper, two policies horizontal and vertical scaling of the resources are considered. Horizontal policies address resource configuration strategies which change the number of active VMs in the system. In contrast, vertical policies are defined at finer grains of control (Elastic VMs) and adjust the amount of allocated resources based on the new demands of the VM. Since the scaling happens online and there is no need to reboot the instance, vertical scaling is much faster and does not adds extra costs for a software license or wasted resources.

Upon receiving an anomaly alert from anomaly prediction module, the framework should create a new record to flag the beginning of a new anomaly event in the target VM. However, we need to consider the transient changes in the system that may cause false alarms. As a result, a new anomaly event is recorded at time t if the current observation is showing an anomaly alert as well as all the past observations in the window $\{t$ - K, t-K-$1, ..., t$-$1\}$. In other words, the system ignores the first K alarms for one VM until there will be at least $K+1$ consecutive alerts notifying an anomalous behaviour. A proper value for K can be selected considering the trade-off between computation overheads, the stability of the environment and the performance degradation tolerance. Small values of K may cause the system to perform unnecessary checks of the performance or decide on preventive actions for many false alarms, while large values of K increases the time it takes for the system to start a scaling action in response to the performance problems.

In the proposed framework, some conditions should be met before resource manager decides on a new scaling action for the system. The following subsections and Algorithm 3 explain these conditions.

Algorithm 3: Vertical Scaling Policy

input : *counter*: Number of recent alerts for the VM

input : *anomalyDetected*: True if recent anomaly score exceeds threshold

input : *cause*: The root cause detected for the current anomaly

Parameter: K: Minimum Number of Alerts to Record an Anomaly

θ: Anomaly Score Threshold

```
if anomalyDetected then
    counter ← counter + 1
else
    counter ← 0
end
if counter > K then
    if system is not in cooling period && cause ≠ Load then
        If system is moving toward critical condition based on Algorithm 4, start a vertical
         scaling action.
    end
end
```

4.3. Per-VM Vertical Scaling Policies

After receiving a confirmed anomaly event for one VM, the VM starts to check if some type of the resource adjustment is required. ACAS considers scaling strategy only when a performance degradation or SLA violation is observed. In this case, we consider the breach of the resource utilization thresholds as a sign of the violation of SLA objectives. Let th_i be the threshold for resource i. If the utilization of this resource at time t is more than th_i, system records a violation of SLA starting from time t. Therefore, no corrective action is triggered if there are enough spare resources to fulfill the requests during next time intervals. One question to be answered here is that what is the best time interval to predict the future resource usages of resources. Since the online resource adjustments in elastic VMs become effective almost immediately, we take one time interval away from the recent observation as the prediction interval. Therefore, the framework sends back the list of all metrics that are predicted to violate their respective thresholds at the next time interval.

Since vertical scaling is a response to local anomalies happening in a VM, the load problem is

Algorithm 4: Identification of System Criticality

input : $D = (X_1^m, X_2^m, ..., X_{lw}^m), X_i^m \in R^{1 \times r}$: A matrix of lw records, each record including measurements for r features

input : *cause*: Root cause detected for current anomaly

Parameter: LI: Log Interval

$delay \leftarrow 0$

if $cause \neq Load$ **then**

  $delay \leftarrow VerticalScalingDelay$

else

  $delay \leftarrow HorizontalScalingDelay$

end

$windowLength \leftarrow delay/LI$

$P = (X^p_{lw+windowLength}) = f_p(X_1^m, X_2^m, ..., X_{lw}^m)$

for *feature* $i \in P$ **do**

  if P_i *exceeds* th_i **then**

    $f_i \leftarrow 1$

  end

end

Decide about the criticality of system based on the information provided by f_i

ignored at this step and local resource adjustment is triggered if the detected problem is related to one of the resource level metrics of the VM. Depending on the metric detected as the root cause of the problem, the system decides about changing the number of CPU cores or the amount of memory capacity of the VM to prevent performance degradations in the application. After starting an auto-scaling process, the VM will enter in a locked state which means that during this time no other scaling action is performed. The reason is that it takes some time for the system to adapt to the changes of the resources, so the first few anomaly alerts are ignored to give the system enough time to reach a stable state.

4.4. Horizontal Scaling Policies

A horizontal scaling policy is performed if there are no VM in the locked state, meaning that there has not been any vertical scaling in the recent intervals that can affect the state of the system. First, the state of all VMs is checked and the number of VMs which are moving toward critical condition is recorded. One VM is recognized in critical condition if at least one of the main attributes is predicted to breach the threshold in the near future. Similar to vertical scaling procedure, we consider a rough estimate of the time it takes to boot a new VM in the system as prediction interval. In other words, ACAS asks for enough time to add a new VM before the system enters the anomaly state. If all the active VMs are found moving toward the violation state, an alert to add a new VM is issued. Afterwards, the system starts a cooling period when no scaling will take place. This waiting time is required so the load balancer can detect new VM and start sending new requests to that.

5. Performance Evaluation

The proposed framework encompasses multiple components from resource monitoring, resource configuration and data analysis. The framework is general and should be applicable to different types of applications and workloads. However, in order to demonstrate the functionality of ACAS, we select web applications which are shown to be prone to many performance problems involving CPU and memory resources [2]. The main focus of this work is the performance of the application layer which can be easily affected by the behaviour of users, buggy codes or other malfunctioning applications.

To validate the framework, we use CloudSim discrete event simulator [7] which is a framework for modeling and simulation of cloud computing infrastructures and have been used extensively for validation of cloud services and applications. An extension of the CloudSim is leveraged that implements an analytical performance model of 3-tier applications in the cloud and multi-cloud environments [8].

Table 3: Experiment Configurations

Variable	Description	Value
K	Minimum number of alerts before an anomaly record is created	6
lw	Number of observations in learning window	60
tw	Number of observations in training window	300
L	Minimum number of violations before baseline approach starts an action	2
LI	Monitoring Interval (Log Time)	60
θ	Anomaly Score Threshold	0.55

5.1. Experimental settings

The experimental environment is simulated as one cloud data center hosting the application and database servers. The application servers are modeled with the initial configuration of one virtual core, 3.75 GB of RAM and Linux operating system. The VM start-up times are modeled based on the performance study done by [9].

The following experiments are based on an extension of CloudSim which models the workloads on Rice University Bidding System (RUBiS) benchmarking environment [10]. RUBIS is a benchmark that implements the core functionality of an auction site including browsing, bidding and selling modeled based on eBay.com. RUBIS follows a 3-tier web based framework consisting of the client, application and database servers. Sessions are the unit of works defined in the RUBIS and represent a sequence of requests from one customer interacting with the application. The resource usage of each session is monitored and modeled in the CloudSim based on the work done by [8]. In total, there are 4 attributes CPU, memory, I/O usages as well as the number of sessions which are collected during each experiment for data analysis part. For the details of how the workload is modeled and validation of the extracted models you can refer to the work [8]. To implement the prediction step, we utilize *forecast* package implemented in R [1] which models a feed-forward neural network with lagged inputs for forecasting univariate time series. The generated network is modeled by one lagged input and one bias node, one hidden layer with one node and one bias node and one output which

[1] https://www.rdocumentation.org/packages/forecast/versions/8.1/topics/nnetar

is analogous to AR model but with a non-linear function.

In order to demonstrate the functionality of ACAS in resolving local performance problems with the help of fine grained resource scaling, we have also extended CloudSim framework to enable the on-the-fly changes of the resource configurations without turning the VM off. Two main resource types CPU and RAM are considered in this implementation. However, the codes are general and can easily be extended for other types of the resources. The amount of changes in each scaling action can be configured to be a percentage of the original capacity of the resource. For the following experiments, the capacity of CPU resources increases by one core (100 percent of initial configuration) while the RAM storage is increased by 20% for each scaling action. Moreover, the anomaly detection models in ACAS are generated using IsolationForest package implemented in R environment[2]. In order to connect the anomaly detection module to the simulation environment which is developed in JAVA, we utilize *Rengine* interface which supports calling R implemented functions from Java environment.

Each experiment has a duration of 18 hours with sessions arrival time modeled as a Poisson distribution with a frequency that is defined as a function of time [8].

In order to evaluate different aspects of the proposed framework, four cases of experiments have been run. In two cases, the behaviour of ACAS is tested in the presence of local anomalies in the VMs. In two other cases, the system is exposed to workload increases and the functionality of the framework is analyzed. Two types of the resource level bottlenecks, insufficient memory and CPU, are simulated. In both cases, we focus on the impact of increasing trends on the corresponding attribute. In order to simulate memory problems in CloudSim, a predefined percentage of the memory storage is removed from the available memory at consecutive interval times which creates an incremental trend in the used memory of the VM. For insufficient CPU, a predefined percentage of the available CPU capacity is flagged as reserved assuming a different CPU-intensive application starts running as a background process along with the target application.

Th idea of performance anomaly detection has been widely investigated in the research area. However, most of them follow supervised approaches or are designed for specific scenarios or focus on data analysis part of the problem without providing the details of an integrated framework for the purpose of resource management. In the other hand, many of the famous public cloud providers such as Amazon [1] leverage a threshold based auto-scaling approach for dynamic scaling of their resources. In the threshold based approaches, system continuously tracks the state of the resources in the system and an anomaly alert is triggered if the utilization of monitored metrics exceeds a predefined threshold. For example, a new machine is added to the system if the CPU utilization is more than 80 percent for five continuous sampling intervals. Therefore, for the comparison purpose, we have implemented the same threshold method as our baseline approach. To have a comparable

[2]https://sourceforge.net/projects/iforest/

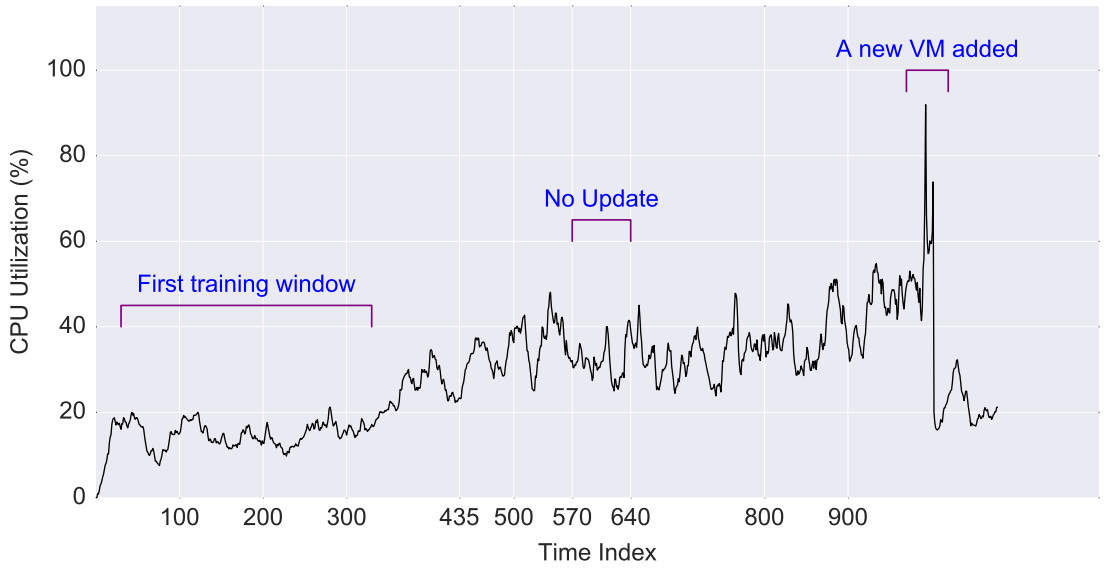


Figure 3: The process of ACAS on a sample workload including the first training window and one horizontal scaling action. One part of the data that is analyzed with the same models (no model update occurred during this time) is also annotated.

experiment, the thresholds for the baseline auto-scaler is the same as the triggering thresholds of ACAS framework. In the all experiments, this value is equal to 70 percent and is similar for both CPU and memory. A cooling period of 15 minutes is considered for the baseline simulation. Therefore, no two auto-scaling are performed in a time interval less than the cooling period. Table 3 shows the values of parameters used in the experiments.

5.2. Experiments and Results

In the first experiment, we investigate the behaviour of ACAS based on a sample workload similar to Figure 3. The experiment starts by sending requests to a load balancer which distributes the load among application servers on a round robin basis. In order to start training the models, we follow the observations from [6] which suggests that 2^8 generally is enough to consider as the sample size (ψ) for training phase. Considering this and based on the nature of the dataset and empirical experiments to apply IForest as an online anomaly detection algorithm, 300 is selected as the training window size (tw) to be considered for the sampling and training purpose. Therefore, the anomaly module waits for the first 330 observations to pass and then initializes the first anomaly detection models by training IForest algorithm with the last 300 records as it is shown on Figure 3. The first 30 records are ignored to let the system a time to reach a stable state meaning that at this time the recorded utilization shows the actual resource usages of the VMs considering their workload.

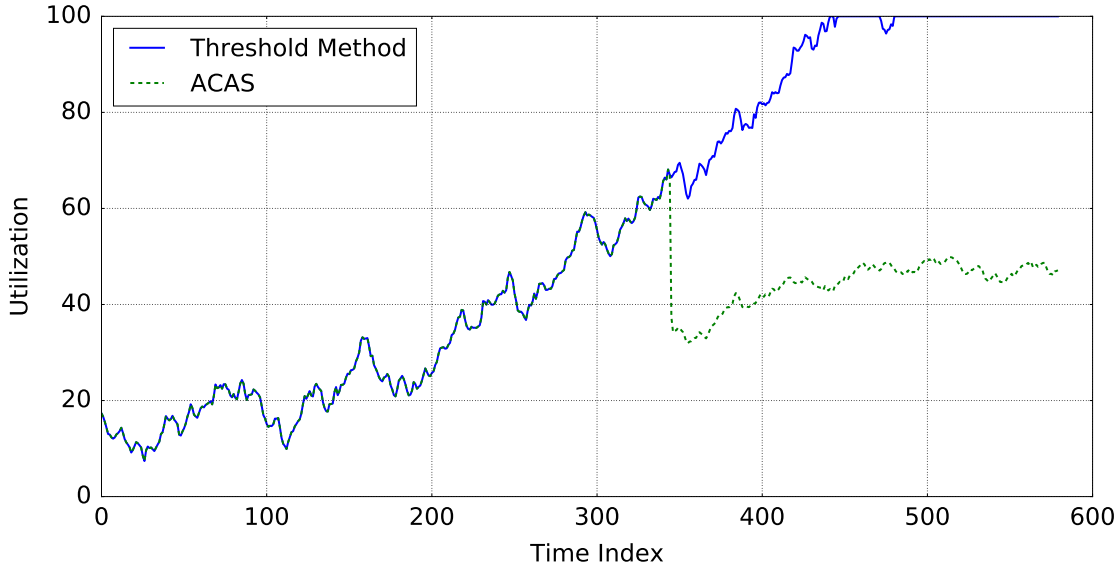
After the first initialization, anomaly detection module starts to regularly check the performance

Table 4: Number of times that resource utilization exceed the threshold before the first auto-scaling action is triggered. *NA* means no scaling is performed.

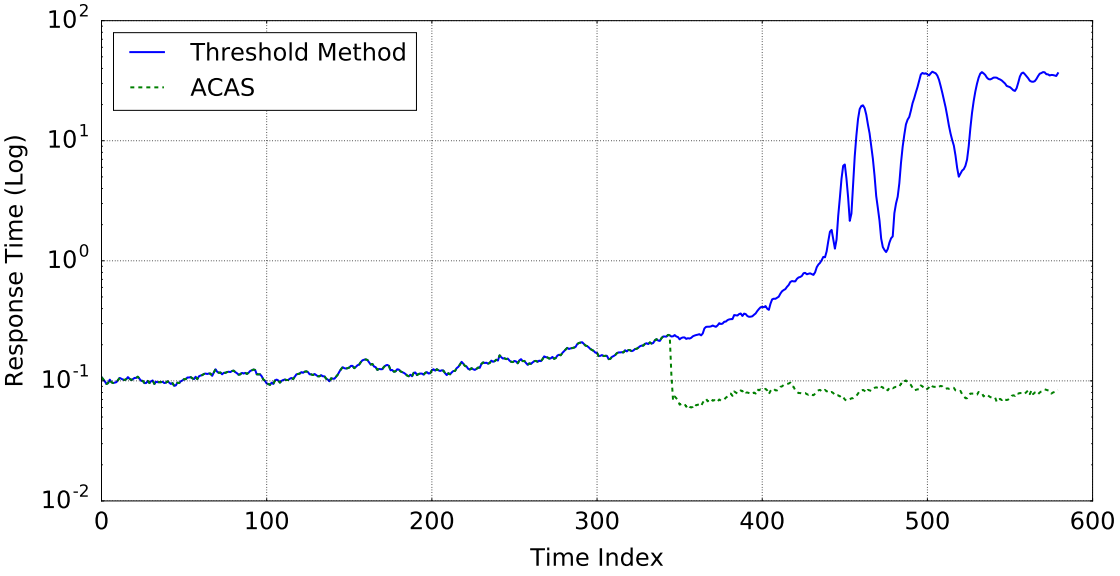
Anomaly Type \ Algorithm	**ACAS**	**Threshold Method**
CPU	1	NA(>100)
Memory	10	NA(>100)
System Load	5	8

of the system by applying the generated anomaly detection models on the recent collected observations at the configured time intervals (presented as a while loop in Algorithm 1). However, depending on the state of the system and based on the definitions discussed in the Section 4.1, models may need to be updated occasionally to represent the new state of the system. As we can see in Figure 3, after the first model initialization, a low rate increase of the incoming load is started which corresponds to a transition state based on our definitions. Therefore, the first update of the models recorded for this experiment is occurring around 435th observation when the system is identified at the end of the transition and entering a new normal state. Similarly, other updates occur occasionally during the experiment due to the fluctuations in the utilization data. However, there are also several gaps that no update has occurred during that times. These gaps are consistent with our observations of the stability of average utilization data and the functionality of ACAS which has not detected any *transition* that requires new model trainings. For example, there is no update between observations 570 to 640 or there are only 7 updates between observations 645 to 760. The reduction in the number of updates helps the system to decrease the overhead of recurrent trainings to create new models. The same procedure with similar reasoning is applicable for the next load increase, starting around observation 800, that changes the state of the system from *normal* to *transition* and also triggers an auto-scaling action which adds a new VM to the system.

The next experiments are designed to test the presence of the local anomalies in VMs. The initial configuration is done by adding 3 application and 2 database servers in the system. Then, one VM is randomly selected as an anomalous VM. For the both experiments of CPU and memory anomaly, we wait for a minimum of 5 hours and then, at a random time, the injection of the anomaly in the VM is started. Table 4 shows the number of recorded observations that the attribute corresponding to the detected root cause exceeds the threshold. *NA* in the table means that there was no auto-scaling action in the response to the injected fault in the target VM which equals to a 100 percent violation of

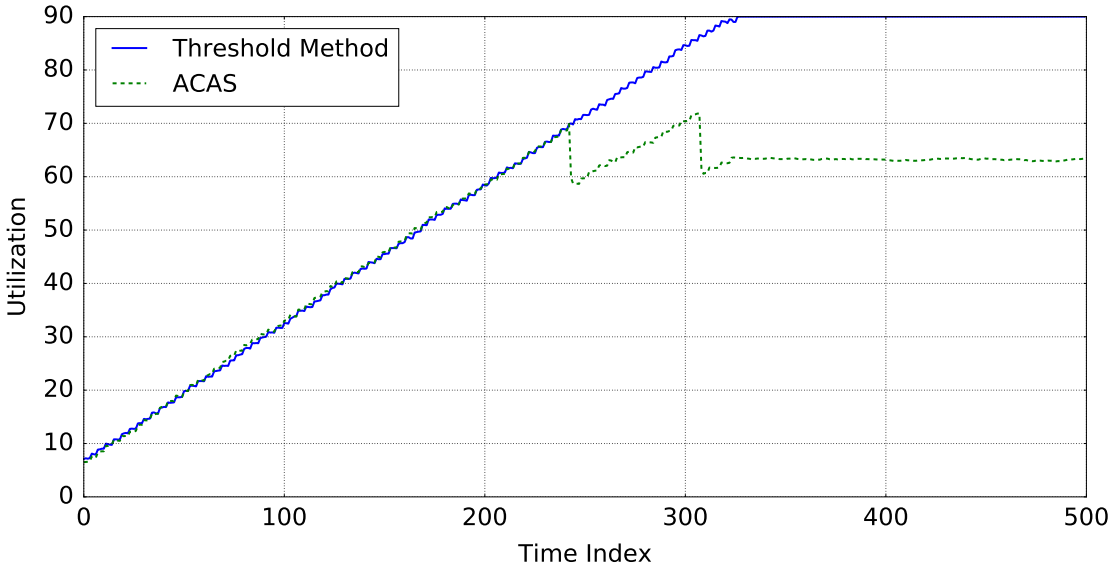


(a) CPU Utilization

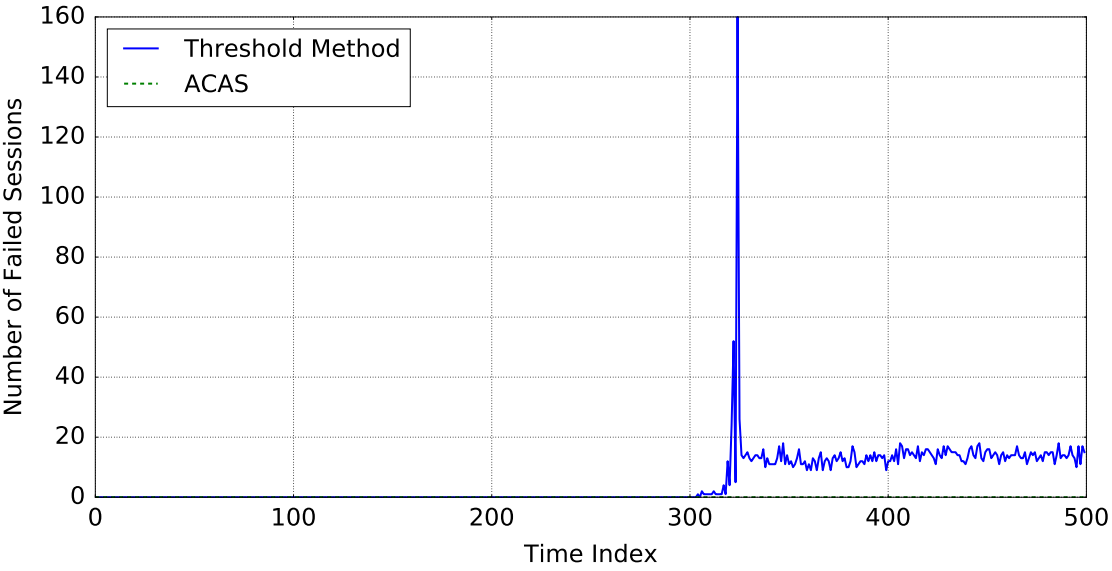


(b) Response Time (Log scale)

Figure 4: Vertical auto-scaling for CPU bottleneck. ACAS avoids high response times by timely reaction to the predicted performance problem.



(a) Memory Utilization



(b) Failed Sessions

Figure 5: Vertical auto-scaling for Memory bottleneck. ACAS avoids failed sessions by timely reaction to predicted performance problem (ACAS line for the failed sessions is zero for duration of the experiment).

the SLA. The exact number of the violations depends on the duration of the corresponding anomaly in the system which may last for hours.

For more clarification of the way these policies are reacting in the presence of local performance problems, Figure 4 and Figure 5 present the utilization of the corresponding attribute for each fault and for the both ACAS and threshold methods. Regarding CPU, the ACAS has increased the number of cores by one as soon as it predicts the criticality of the CPU utilization measurements. One violation is observed in this case which is a result of the fast changes in the attribute values which the prediction function has not caught. In contrast, the threshold approach is monitoring the average state of the whole system, missing the local faults occurring at the anomalous VM. It's worth mentioning that even having a per-VM monitoring mechanism for the threshold approach can only help to trigger a horizontal scaling with the condition that the monitored values show a minimum of L violations before auto-scaler starts triggering an action. The L value should be chosen reasonably to avoid unnecessary scalings in the presence of temporal changes in the system. In our simulations, L has a small value equal to 2. However, depending on the application instability, this value can be higher which leads to even more violations. This situation is a result of the lack of the knowledge about preceding trends to the anomaly state. ACAS solves this problem by keeping the track of the patterns in the data and performing the scaling when the conditions of being in a continuous anomaly state and the violation of the threshold values are met.

The above reasoning is also applicable for memory bottlenecks. One point to mention is that the simulated RUBIS application shows a CPU intensive behaviour. Therefore, memory usage has less fluctuations and shows more clear change points which can be detected with higher accuracy. Figure 5 shows two sequential vertical scalings of memory which adds 20 percent of the initial capacity each time. The first scaling happens before any violation is observed which shows the prediction part of ACAS helps the scaler to perform a proactive action to predict the future anomaly events and start a corrective action. The results show that the memory usage drops down by 20 percent. However, the utilization continues to increase which causes the start of the second scaling action. This time, however, a few numbers of violations for the memory usage are observed. The reason is that for a small duration of time after the first scaling, the system is recognized in a new *changed* state which is followed by an update of the models. Therefore, the initial increases in the memory do not trigger anomaly alerts which cause the system to start the second action after some delays. In this case, the reactive part of the approach helps the system to detect the anomaly state when the violations are observed.

Figure 5 also shows the number of failed sessions for the both policies. A session is flagged as failed if the VM does not have enough memory to process its requests. As the figure shows, in the experiments with the threshold method, the number of failed sessions has increased as a result of ignoring the local fault in the VM. In contrast, ACAS has properly adjusted the configurations

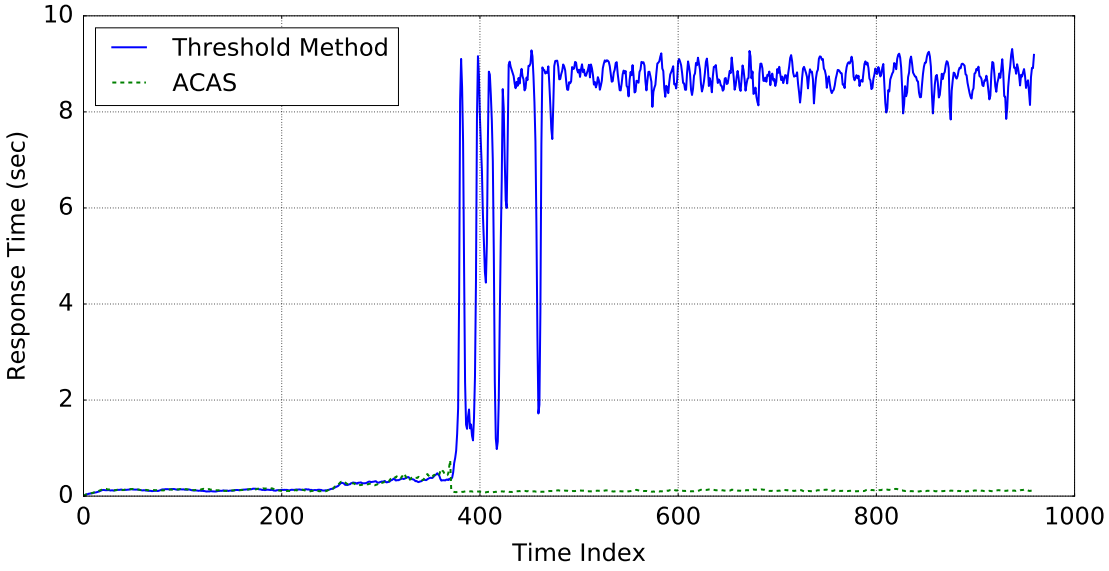


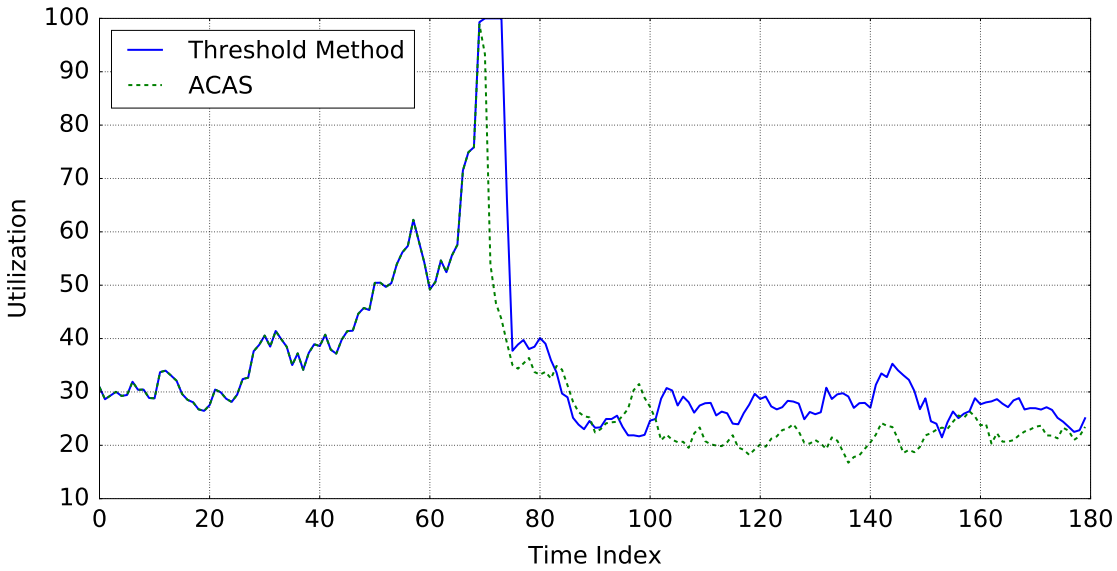
Figure 6: Response time of one application server when the machine is overloaded

corresponding to the bottleneck resource which avoids the unusual increases in the failed sessions.

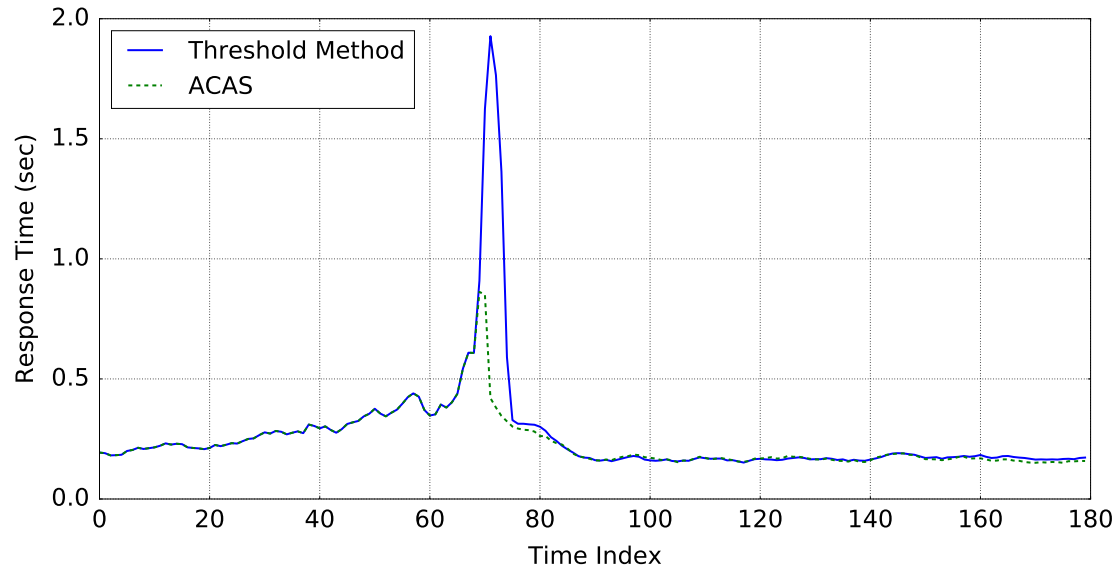
As demonstrated by aforementioned experiments, the proactive vertical scaling helps to quickly target the bottleneck resource and reduce the number of violations by adjusting the amount of resources accordingly. This process also helps to reduce the cost as well as the energy consumptions compared to the conventional way of adding new VM machines in the system. It is also worth noting that the local execution of anomaly detection reduces the complexity of training the anomaly detection models to one VM. As it is explained in Section 3 the time and space complexity of IForest algorithm is constant when the same number of training observations is used for model generation.

The next set of the experiments analyzes the behaviour of the system when the input workload of the machines suddenly increases. Two types of the problem have been considered. The first experiment simulates an environment where one VM is exposed to an increasing workload while other VMs in the system stay in their normal state. Therefore, one VM is randomly selected and the number of the requests sent to this VM is increased. This scenario can happen in different cases such as a result of a misconfigured balancer service which assigns a higher weight to one VM. Figure 6 shows the impact of the load increase on the response time of the target VM for both policies. As we expect, the threshold approach is not successful at detecting the local performance problem and many violations of the response time are observed. In contrast, the local anomaly detection approach utilized by ACAS helps to identify the problem as soon as the metrics show an increasing trend followed by exceeding the thresholds.

The second experiment for the load problem simulates an overloaded system where the number of incoming requests to the balancer is increased, resulting in the increase in the resource usage of every machine at the same time. This scenario is a common case in the web applications known



(a) CPU Utilization



(b) Response Time

Figure 7: CPU Utilization and Response Time of one application server when the system is overloaded. ACAS is able to proactively trigger a horizontal scaling action compared to reactive response of the threshold method which causes more SLA violations.

as flash crowds when sudden surges in the traffic to a web site causes high delays in the response time making it virtually unreachable for the users. As Figure 7 shows, both policies make similar decisions and add a new VM after the problem is recognized. However, ACAS approach is able to react to the problem immediately at the same time that the first breach of the threshold is detected which causes the system to return back to the normal state after 5 observations of the violation of CPU and memory metrics. In contrast, the threshold approach does not have a knowledge of the past behaviour of the system and therefore delays the triggering of the auto-scaling action for L observations. In our experiments, this value is set equal to 2 which results in about 8 violations before the system goes back to the normal state. Larger values for L, more SLA violations in the system.

Finally, a set of the plots presenting the relation between anomaly scores and model updates are shown for a sample experiment in ACAS framework. Figure 8 shows the CPU utilization of one application server. The marked points are the observations recognized as anomalies, meaning that the corresponding anomaly scores are higher than 0.55. Figure 9 presents a combined view of the anomaly detection process for the same workload, including detected anomaly points along with the anomaly update times. Each point at the top line shows that the observation at the corresponding time was detected as an anomaly, while the gaps between these point reflect normal or transition states of the system. The first 330 points are ignored as they are used during training phase and detection process was not activated at that time. Similarly, each point at the bottom line shows that a model update happened at the time of the corresponding observation. As you can see, at the times that the system is recognized in the normal state, no update is occurring, meaning that the models are reflecting the current state of the system. Another observation from these figures shows that the updates are delayed when an anomaly event is started while the system is recognized as being in the transition state. An example of this condition can be seen between observation 900 to 1100 which is reflected by the gaps among the points at the bottom line.

6. Conclusions and Future Work

Elastic VMs with the accompanying knowledge from analyzing performance data can bring new opportunities to offer better resource management solutions in the distributed environment. In this work, we show how fine grained resource configurations can help to improve the auto-scaling solutions for a category of local anomalies occurring in one VM. The proposed ACAS framework utilizes a low overhead anomaly detection solution based on the IsolationTrees and combines it with a cause identification procedure to enable a proper auto-scaling solution considering various types of anomalies in the system.

For the future work, we plan to improve the cause detection method with the aim of detecting more complex anomaly problems, the ones that can affect the pattern of the data in long run

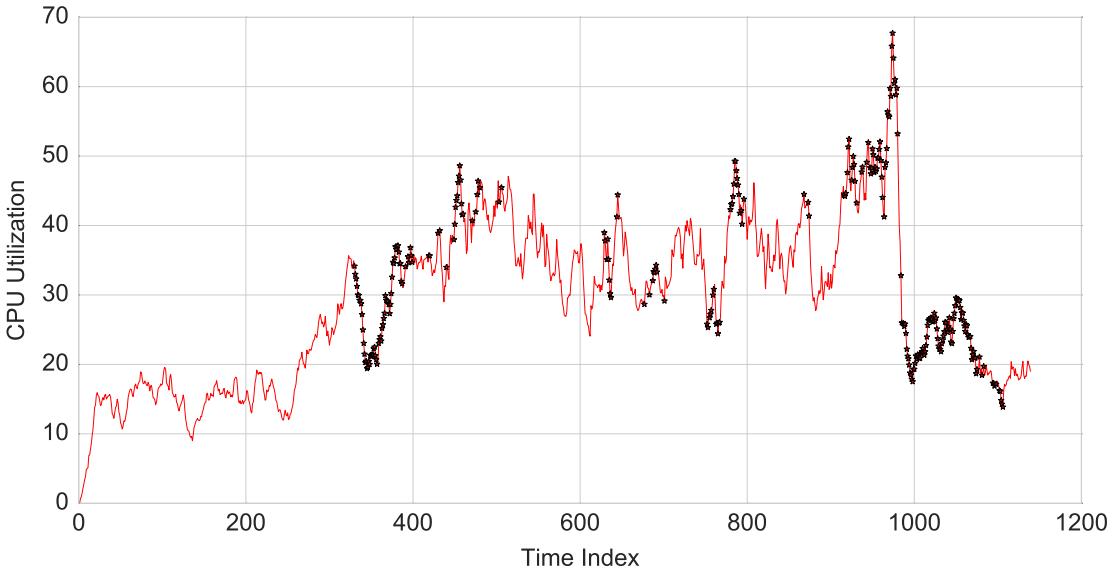


Figure 8: CPU utilization of one application server when the machine is overloaded. The marked points are the records detected as anomaly.

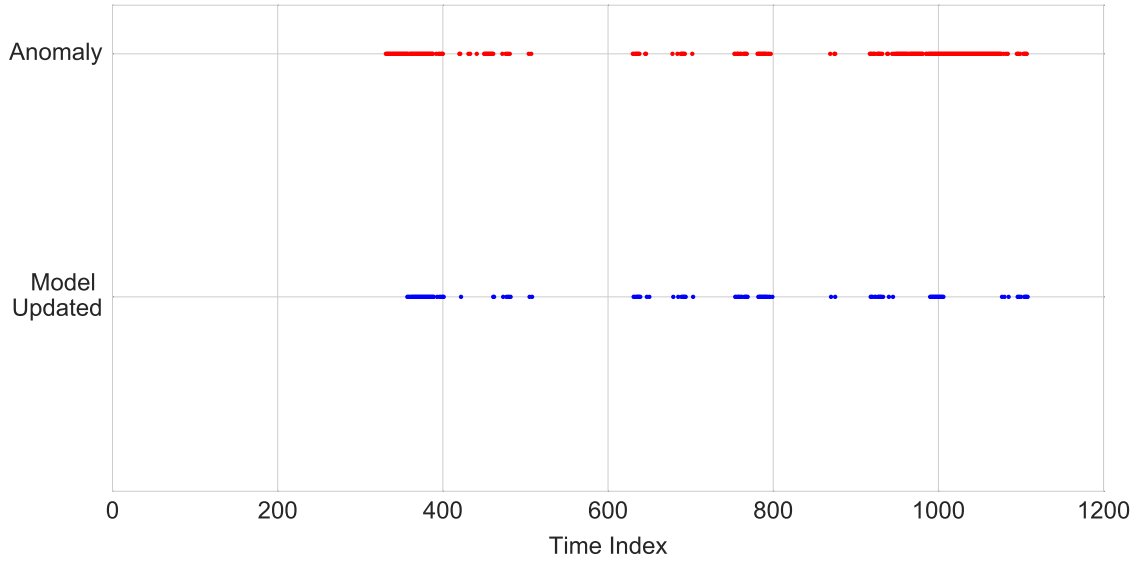


Figure 9: Detected anomaly points and the model update times for the duration of the experiment. Red points show the observations that detected as an anomaly. Blue points show the times that a model update occurred in the system.

or influence multiple attributes. As an example of more complex types of anomalies, distributed attacks on the cloud resources are a challenging area to be investigated, especially considering the other sources of data such as packet information which should be included along with the previously discussed resource level measurements. In this case, we may need to add a profiling step to record a signature of old and new anomalies, categorizing them based on the underlying reason of the problem. Moreover, we can work to further improve the automatically of the framework by adjusting the thresholds for both anomaly scores and attribute dependent violations based on the type of the anomaly, the intensity of changes in the data, etc. Finally, we can extend our work to consider alternative virtualization methods such as containers in which the same Operating System (OS) is shared among multiple container-hosted applications. While the proposed ACAS framework is applicable to these environments and we expect similar performance of our framework when applied to containers at the resource level, the nature of containers may introduce some additional requirements in terms of measuring resource utilization with respect to each container. Therefore, the isolation of performance may become more challenging which requires finer levels of data analysis such as monitoring system call traces and interactions among containers to alleviate corresponding performance degradations.

[1] Amazon, Amazon 2018.
URL `https://aws.amazon.com/`

[2] B. Subraya, Integrated Approach to Web Performance Testing: A Practitioner's Guide, IGI Global, 2006.

[3] B. Hong, F. Peng, B. Deng, Y. Hu, D. Wang, Dac-hmm: Detecting anomaly in cloud systems with hidden markov models, Concurrency and Computation:Practice and Experience 27 (18) (2015) 5749–5764.

[4] C. A. Cunha, L. Moura e Silva, Separating performance anomalies from workload-explained failures in streaming servers, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012), CCGRID '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 292–299.

[5] Q. Guan, S. Fu, Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures, in: Proceedings of the 32Nd IEEE International Symposium on Reliable Distributed Systems, SRDS '13, IEEE Computer Society, Braga, Portugal, 2013, pp. 205–214.

[6] F. T. Liu, K. M. Ting, Z.-H. Zhou, Isolation-based anomaly detection, Vol. 6, ACM Transactions on Knowledge Discovery from Data, New York, NY, USA, 2012, pp. 3:1–3:39.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, Cloudsim: A toolkit

for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exper. 41 (1) (2011) 23–50.

[8] N. Grozev, R. Buyya, Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments, The Computer Journal 58 (1) (2013) 1–22.

[9] M. Mao, M. Humphrey, A performance study on the vm startup time in the cloud, in: Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, CLOUD '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 423–430.

[10] RUBIS, Rice university bidding system.
URL `http://rubis.ow2.org/`

[11] R. N. Calheiros, K. Ramamohanarao, R. Buyya, C. Leckie, S. Versteeg, On the effectiveness of isolation-based anomaly detection in cloud data centers, Concurrency and Computation: Practice and Experience (2017) e4169.

[12] S. Di, D. Kondo, W. Cirne, Google hostload prediction based on bayesian model with optimized feature combination, Journal of Parallel and Distributed Computing 74 (1) (2014) 1820–1832.

[13] K. Cetinski, M. B. Juric, Ame-wpc: Advanced model for efficient workload prediction in the cloud, Journal of Network and Computer Applications 55 (2015) 191–201.

[14] J. Yang, C. Liu, Y. Shang, Z. Mao, J. Chen, Workload predicting-based automatic scaling in service clouds, in: Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, CLOUD '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 810–815.

[15] L. Yazdanov, C. Fetzer, Vscaler: Autonomic virtual machine scaling, in: Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, CLOUD '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 212–219.

[16] D. Grimaldi, V. Persico, A. Pescape, A. Salvi, S. Santini, A feedback-control approach for resource management in public clouds, in: IEEE Global Communications Conference, 2015, pp. 1–7.

[17] J. V. Bibal Benifa, D. Dejey, Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment, Mobile Networks and Applications.

[18] J. . Cid-Fuentes, C. Szabo, K. Falkner, Online behavior identification in distributed systems, in: 34th IEEE Symposium on Reliable Distributed Systems, Montreal, Quebec, Canada, 2015, pp. 202–211.

[19] X. Gu, H. Wang, Online anomaly prediction for robust cluster systems, in: Proceedings of the 25th IEEE International Conference on Data Engineering, 2009, IEEE, Shanghai, China, 2009, pp. 1000–1011.

[20] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, D. Rajan, Prepare: Predictive performance anomaly prevention for virtualized cloud systems, in: Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems, Macau, China, 2012, pp. 285–294.

[21] R. Han, L. Guo, M. M. Ghanem, Y. Guo, Lightweight resource scaling for cloud applications, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012), CCGRID '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 644–651.

[22] G. Moltó, M. Caballer, C. de Alfonso, Automatic memory-based vertical elasticity and oversubscription on cloud platforms, Future Gener. Comput. Syst. 56 (C) (2016) 1–10.

[23] D. J. Dean, H. Nguyen, X. Gu, Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems, in: Proceedings of the 9th International Conference on Autonomic Computing, ACM, New York, NY, USA, 2012, pp. 191–200.