Author/s:
Kardani-Moghaddam, S;Buyya, R;Ramamohanarao, K

Title:
Performance anomaly detection using isolation-trees in heterogeneous workloads of web applications in computing clouds

Date:
2019-10-25

Citation:
Kardani-Moghaddam, S., Buyya, R. & Ramamohanarao, K. (2019). Performance anomaly detection using isolation-trees in heterogeneous workloads of web applications in computing clouds. Concurrency and Computation: Practice and Experience, 31 (20), https://doi.org/10.1002/cpe.5306.

Persistent Link:
https://hdl.handle.net/11343/285752

Performance Anomaly Detection Using Isolation-Trees in Heterogeneous Workloads of Web Applications in Computing 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 is a model for on-demand access to shared resources based on the pay-per-use policy. In order to efficiently manage the resources, a continuous analysis of the operational state of the system is required to be able to detect the performance degradations and malfunctioned resources as soon as possible. Every change in the workload, hardware condition or software code, can change the state of the system from normal to abnormal which causes the performance and quality of service degradations. These changes or anomalies vary from a simple gradual increase in the load to flash crowds, hardware faults, software bugs, etc. In this paper, we propose Isolation-Forest based anomaly detection (IFAD) framework based on the unsupervised Isolation technique for anomaly detection in a multi-attribute space of performance indicators for web-based applications. Unsupervised nature of the algorithm and its fast execution make this algorithm most suitable for the environments with dynamic nature where the patterns of data change frequently. The experiment results demonstrate that IFAD can achieve good detection accuracy especially in terms of precision for multiple types of the anomaly. Moreover, we show the importance of validating the accuracy of anomaly detection algorithms with regard to both Area Under the Curve (AUC) and Precision-Recall AUC (PRAUC) in an extensive set of comparisons including multiple unsupervised algorithms. The demonstration of the effectiveness of each algorithm shown by PRAUC results indicates the importance of PRAUC in selecting suitable anomaly detection algorithm which is largely ignored in the literature.

Keywords: Cloud computing, Anomaly Detection, Isolation Trees, Resource Management

1. Introduction

In recent years, the emergence of cloud service providers (CSPs) such as Amazon, Google and Microsoft has moved the previously limited, community specific capabilities of high performance computing to a new era of public, on-demand, pay-as-you-go computing. These new characteristics offered by cloud providers mainly highlight the need for more complex and robust resource management solutions that decrease the need for human involvement. The main goal for CSPs is to find better ways of resource management to

*Corresponding author s.moghadam@live.com

improve resource utilization and guarantee the quality of service (QoS) experienced by their customers. Any violation of these service level agreements (SLA) can cost providers penalties for SLA violations or losing their reputation. However, considering the dynamic nature of cloud systems, every change in the workload, hardware condition or software code, can change the state of the whole system from normal to a state of abnormal behavior which can affect the performance and QoS. The degradations in the performance can result in higher monetary costs and energy wastage for under-utilized resources which are negative sides of the dynamic environment from resource provider perspective, highlighting the need for automated ways of detecting performance problems [1]. This is a highly important observation, especially for large scale web application systems where the interaction from users to web servers can change frequently, affecting the pattern of workloads and resource requirements. For example, it is shown that the web applications are prone to many of the performance problems which involves CPU and memory resources [2].

Taking into account that each type of performance problem can impact the system or application metrics differently, defining proper rules that cover all types of problems is becoming complex and out of the expected knowledge of application owners. It is vital for every resource management solution to consider utilizing timely and adaptive algorithms to identify the anomalies in the system as soon as possible. Therefore, researchers are looking for more powerful solutions for performance analysis of resources in the cloud. An automatic anomaly detection module should be able to analyze the collected performance metrics from cloud resources and build models which can detect deviation points where the system moves to an anomaly state. In the process of collecting metrics, building models and triggering alerts, there are some challenges that should be considered:

- **Scalability**: One of the main characteristics of cloud dependent applications is scalability which makes it possible to scale up the system components to hundreds and thousands of virtual machines (VMs). In such a dynamic environment, centralized anomaly detection approach becomes a problem especially if we want to capture the state of the whole system in one model. As a solution for this problem, we assume that each machine monitors the performance metrics of its own VMs which breaks down the problem of anomaly detection to one host. Furthermore, we are utilizing an easy to deploy monitoring tool, known as Ganglia that can be easily managed in a large distributed environment.

- **Unsupervised learning**: Cloud environment is prone to different types of anomalies that affect the performance of the system in different ways. Therefore, it is reasonable to assume that we do not have access to labels that identify the state of the system as normal or anomaly. Accordingly, the proposed anomaly detection module does not assume prior knowledge of the system and would perform in an *unsupervised* manner.

- **Recurrent model parameters tunning**: Due to the heterogeneous nature of web applications in the cloud, the normal state of the system can change significantly based on the number of requests sent to the system. In this case, detecting anomalies in a previously unseen normal environment is another

challenge that we should consider. Most of the existing algorithms require tunning and parameter settings to be done before updating the models. This procedure adds extra overhead on the system, particularly for frequently changing environments. The proposed anomaly detection approach is fast which requires no workload dependent configuration or any time consuming data preparation which makes it a fit for our target environment.

- **Application preferences**: The problem of anomaly detection is highly application and data dependent. We need to consider cases when the application owner prefers an algorithm with a higher precision, sacrificing the sensitivity of the algorithm or vice versa. For example, applications concerning disk drive failure analysis or medical tests for rare diseases require low and more controlled false alarms rates considering the costs of triggered actions for predicted anomalies [3]. Hence, in this work, we evaluate the effectiveness of proposed IFAD framework with different algorithms in terms of both measures of AUC (Area Under the Curve) and PRAUC (Precision-Recall AUC) and the trade-off between false negative and positive rates in the results. The study helps to understand better the capabilities of algorithms from the perspectives that are usually ignored in current research.

With regard to these challenges, we propose an unsupervised Isolation based anomaly detection framework to detect different types of performance anomalies in 3-tier web-based applications. The **contributions** of this paper are as follows:

- Proposed Isolation based anomaly detection (IFAD): An unsupervised anomaly detection module to detect performance anomalies in web-based applications,
- Deployed a realistic prototype for Web2.0 applications based on the CloudStone benchmark,
- Developed an anomaly injection module by implementing five types of performance anomalies in cloud environments, and
- Demonstrated the impact of changes in the underlying workload on the detection accuracy of algorithms. We measure AUC and PRAUC and their importance for selecting suitable algorithms through comprehensive experiments which give new insights into the importance of application requirements for selecting a proper anomaly detection algorithm. Moreover, the results of comparison with multiple unsupervised algorithms demonstrate the performance benefits of IForest, especially in terms of the PRAUC metric in most datasets.

To the best of our knowledge, this is the first work to investigate an unsupervised anomaly detection approach for analyzing different types of anomalies (CPU, memory, disk, ...) in heterogeneous workloads of web applications in the cloud. Especially, through our experiments, we show that analyzing the performance results by comparing both metrics AUC and PRAUC, which demonstrate the functionality of the algorithms from different points of view, is an important part of the anomaly detection problem that should be further

investigated. Moreover, interesting characteristics of Isolation-Trees based anomaly detection to offer a low overhead algorithm with a simple yet effective procedure makes it a new alternative to analyze other types of anomalies or applications.

The rest of this paper is organized as follows: Section 2 presents the motivation and an overview of the main parts of IFAD framework. In Section 3, we detail the functionality of each part including characteristics of the collected data followed by data processing and finally anomaly detection module. The details of all experiments and the results are presented in Section 4. Section 5 introduces some of the existing works in the field of performance management and anomaly detection and finally, we conclude with the future works in Section 6.

2. Motivation and System Overview

The virtualized environment of cloud models is prone to various types of performance problems that make the resource management solutions more complex and challenging. Any change in the configurations of the resources is a response to a performance degradation or SLA violation which is influencing one or a group of related VMs hosting the application. This is an important issue, especially for web applications in which the behavior of users plays an important part in the performance of the application. For example, the sudden surges in the number of users due to the release of a new product requires an immediate response by adding enough resources to handle the requests. Resource management modules should be aware of these violations to take corrective actions immediately. On the other hand, many existing anomaly detection techniques are not optimized for the highly changing conditions of the heterogeneous applications hosted on the cloud. A large number of clustering and classification techniques are based on the notions of distance and neighborhood density of instances in the feature space of the problem which leads to high computational complexity and memory requirements. The problem becomes more important in the area of cloud performance analysis, where a range of performance indicators from application and system levels can be affected by various types of faults in the system. The more features to be included during analysis, the more complex feature space with higher dimensionality. Moreover, the primary functionality of these algorithms usually targets other problems such as clustering and classification and anomalies are detected as a by-product of data categorization while Isolation technique directly addresses the anomaly detection problem [4, 5].

Regarding the problem of web performance analysis in the cloud, heterogeneity of workloads and different types of possible failures are two contributing factors that can make the above mentioned challenges even more important. Therefore, the first problem is identified as providing a framework that can quickly and effectively model different patterns of workload variations which include a signature of some of the common bottleneck problems. The second addressed problem in this work is the lack of comprehensive study on the effectiveness of various algorithms with regard to the precision of detection results. We have observed that the most common evaluation metrics discussed in the literature focus on the detection accuracy in terms of the true positive points. However, the results of corresponding metrics can be misinterpreted in the problems

Figure 1: 3-tier Web Layers

with highly imbalanced classes which highlights the need for other types of evaluation.

Motivated by the above limitations, we formulate the problem as detecting unusual behaviors that may be a sign of the beginning of an abnormal state in the system with the possibility of changes in the normal patterns of the workload. We show how data analytics techniques can help to overcome the complexities of time series to better capture the pattern of data and anomalies. Regarding this definition, it should be clear what types of abnormal behavior are targeted. Given one VM measurement, we focus on a category of performance problems known as the resource bottlenecks. We try to find point anomalies which are measured records that are not consistent with the previously seen measurements of performance indicators of the system. As an example, consider a situation in which a memory-intensive background application starts working in the same VM with the target application. System observes an increase in the memory related metrics while the incoming workload of the application does not show any significant change. The unusual increase of the memory should be detected as an abnormal behavior as it can cause a performance degradation in the application due to the lack of enough available memory for handling user requests. Similarly, we consider the impact of the unusual increases in CPU, disk and load of the system. It should be noted that the primary focus of this work is to detect unexpected behaviors in terms of the variations of system performance. Therefore, the abnormalities in the resource utilizations are considered as the signs of application performance degradations which is consistent with the previous observations [2]. Moreover, our work concerns a data centric analysis of performance anomalies which do not rely on user perceived measurements such as response time and considers the violations in input data (collected metrics) as possible signs of misbehaving in the system which causes user side SLA violations. This is a common approach in studies that investigate the performance of various anomaly detection techniques in cloud systems [6].

Accordingly, two main parts of the IFAD framework are identified as data preparation modules for each VM to analyze the specific characteristic of the time dependent datasets including trends/seasonality, and the anomaly detection modules. The target application in our model is 3-tier web-based applications which is a standard architecture employed by many web applications [7]. As shown in Figure 1, in this architecture, the functionality of the application is divided into 3 layers:

- Presentation Layer: The interface that user interacts to access the application layer.
- Application Layer: The main parts of the application/business logic are implemented in this layer.
- Data Layer: The persisted data that can be accessed by application layer is manged by one or more

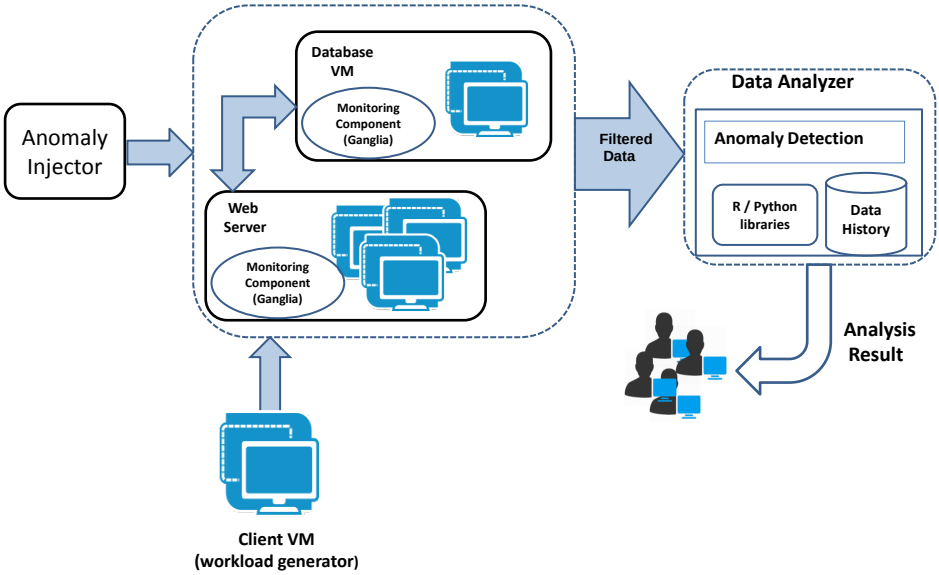


Figure 2: A High Level System Model

database servers in this layer.

It is shown that this type of applications is prone to many of the above mentioned performance problems [2]. The main focus of this work is the performance of the server side VMs which can be easily affected by the behavior of users, buggy codes or other malfunctioning applications.

Figure 2 depicts a high level model of the proposed framework. In order to be able to investigate the effect of different performance changes in the system, a prototype in a virtualized environment is deployed. The prototype models the components of 3-tier applications which are common, yet simple to be manipulated for anomaly injection parts. A set of workloads with different configurations is generated to simulate normal state of the system. Consequently, different types of anomalies are injected into the benchmark components to show how each type of the anomaly can change the state of the system from normal to abnormal.

The VM monitoring module continuously tracks the performance indicators on the VM that is hosting the application. Every VM monitors its own performance metrics at regular intervals and sends them back to the anomaly detection module. The per-VM monitoring policy enables the system to quickly isolate the source of the problem to the target VM and focus on the identification of underlying reason in the machine. This type of problems may be ignored by system-wide solutions that monitor the average values of performance indicators in the system.

The collected metrics are either system related such as CPU, free disk and memory or application specific metrics such as SQL database attributes. Table 1 presents a list of some of the major attributes collected

Table 1: Some of the monitored metrics in the Application or Database servers. In total, there are 98 metrics collected from the monitored machines.

Metrics	**Description**
CPU	CPU utilization
CPU_IO	The time waiting for IO operations
MEM_FREE	Available memory
MEM_DIRT	Memory waiting to be written to disk
$DISK\_READ$	The time in seconds spent reading
$DISK\_WRITE$	The time in seconds spent writing
$BYTES\_IN$	Number of bytes received
$BYTES\_OUT$	Number of bytes sent
$ABORTED\_CONNECTS$	Failed connection attempts to MySQL
$THREAD\_CONNECTED$	Currently open connections to MySQL

in our system. One point worth mentioning here is how to select a proper logging interval for monitoring modules. If the VM records the performance indicators of the system in small intervals, there is a better chance to capture all the patterns and fluctuations in data, which results in faster detection of performance problems. However, a small logging interval means that more data are collected in a fixed time interval which increases the required storage capacity for storing all the values as well as higher overhead at data collection and preparation steps. It is clear that selecting a larger time interval decreases the volume of collected data. However, the system may miss some parts of the changes in the dataset or have a delayed detection of the performance problems, which increases the possibility of SLA violations. Considering these challenges, the selection of a proper log time is highly dependent on the nature of the workload, type of the application and reliability of the environment. Existing works select a range of values from a few seconds to hour(s) based on the target application and problem [6, 8, 9]. In our case, Ganglia gives us the flexibility to select any interval and based on our experiments we have used a configuration of 15 seconds interval to collect the values for all monitored attributes to have a desirable trade-off between storage and accuracy in the system. Collected measurements are processed by different filtering techniques before they are sent to the anomaly analysis module.

The aim of monitoring performance indicators is to detect any abnormality compared to past observations. Upon receiving the filtered measurements for each VM, the system needs to have an abstract model of the past behavior as a reference for the comparison. Anomaly detection module has the responsibility to initialize and update the models by training target anomaly detection algorithm with the past observations. Therefore, the core part of this module is an algorithm which generates the models by learning from training data and applies

them for detecting anomalies in the test data. IFAD leverages isolation based approach for this problem which is detailed in Section 3.4. The output of this module is the anomaly scores for new observations. One can use the obtained knowledge from anomaly detection module to improve decision making procedure of resource management frameworks to start proper mitigation actions or alleviate performance degradations in the system. This can be done by various preventive actions such as changing resource configurations or scaling resources; the details of these techniques however, are out of the scope of this work.

3. System Design

In this section, we explain the data collection and preparation procedure followed by the details of the anomaly detection module.

3.1. Data Collection

In order to have a better understanding of current state of the system and possible drifts to abnormal states, two sources of data are considered: 1) Application data which is workload and component related 2) System data which represents the current state of the resources by combining different metrics such as CPU utilization, memory and disk usage. However, there is no public dataset that provides all the required measurements from application characteristics to VMs performance states. To be able to validate the anomaly detection procedure we need to have a labeled dataset which shows the exact points where an anomaly occurs. Moreover, the source of anomalies can be different. Since we want to have a signature of each type of the anomaly, different anomalies such as CPU or memory bottlenecks should be injected randomly in time in the system. To address these issues, the analysis is performed by generating typical workloads through a standard web benchmark based on CloudStone framework [10]. Each component is installed on a separate VM and the proposed framework is deployed on Australian virtualized cloud infrastructure named Nectar.[1]

To collect the performance indicators of the components, Ganglia framework which is a scalable, distributed monitoring system is utilized [2]. Ganglia is based on a hierarchical architecture with a multicast-based send/receive protocol to monitor the state of the machines in a cluster and aggregates the monitored states in a few representative nodes. The robust design of the data structures and algorithms helps us to easily extend the functionality of the framework by adding new scripts to collect customized measurements from monitored components. For example, the RAM related metrics provided by Ganglia does not include the *active memory* metric. *Active memory* shows the amount of memory recently used considering the fact that some parts of the allocated memory which are included in other metrics such as *consumed memory* are not being used by the application and are set to be freed. Therefore, *active memory* is a more accurate

[1] https://nectar.org.au/research-cloud/
[2] http://ganglia.info/

Table 2: The range of CPU utilization for each workload level

Workload Level	CPU Utilization
Low	10% - 40%
Medium	40% - 60%
High	60% - 100%

estimation of RAM utilization of the application in each time interval. We have extended the basic monitoring module of Ganglia by adding the scripts to calculate this value in our installation. Table 1 shows a list of some of the major attributes collected in our system. The framework collects data in RRD (Round Robin Dataset) format and sends the collected files to the analyzer module. Data analyzer module can read monitored performance data from these files and perform data preprocessing steps before applying the detection algorithms.

As we already stated, the increase/decrease in the number of users interacting with web application can highly impact the pattern of the workload and the utilization of the resources. In order to have a comprehensive validation of the effect of possible trends in the experiments, five types of the dataset are generated by changing the frequency of increase in the number of concurrent users in the system. The changes in the number of users can happen at the start of each step which corresponds to one run of the benchmark. For reproducibility of data by other researchers, we annotate each dataset with the level of resource consumption from the starting point to the end. Having these annotations, we can regenerate the datasets on machines with different specifications. We define three levels of the number of users based on the observed resource utilization during different experiments on the benchmark. Since the target workloads are more CPU intensive, we correspond each level to a range of CPU utilization for the application server. Table. 2 shows three ranges of CPU utilization for these levels. The details for each dataset are as follows:

Dataset1: The number of concurrent users in the system is medium. Therefore none of the resources is overloaded and there is plenty of free CPU and Memory space. The frequency of changes in the number of users is very low, so it simulates a workload without any load related fluctuations, and anomalies show a distinguishable pattern in all parts of the data.

Dataset2: The number of concurrent users in the system is very high. Therefore the utilization and fluctuations of resources especially for CPU are high, which makes it hard to get a well separated pattern of anomalies. The changes in the number of users are very low, so it simulates a workload without any recognizable trend.

Dataset3: We start to increase the concurrent number of users from a low level to a medium level which creates a visible trend in the number of users as well as resource utilization. The increase is performed by adding 10 users every 10 steps. However, due to medium level of resource utilization, anomalies still show distinctive patterns in the attribute space.

Dataset4: We start to increase the number of concurrent users from a low level to a very high level. Therefore it simulates a fast-changing workload sent to the web server and causes higher utilization and fluctuations compared to dataset3. The increase is performed by adding about 10 users every 7 steps.

Dataset5: We start to increase the concurrent number of users from a low level to medium level by adding 10 users every 5 steps. Due to the high rate of increase in request numbers, the noise from high fluctuations is affecting all parts of the data.

CloudStone helps to generate these workloads with dynamic characteristics of web loads, considering various patterns in terms of seasonality and trends in a stream of requests. Therefore, training and testing can be done on consecutive windows of time series as described in Section 4.

3.2. Data Preparation

When applying the IForest to learn from training data with the injected anomalies, we found that a combination of multiple data transformations is required to improve the detection efficiency of algorithms. First, all the features with constant variance are removed as they usually do not provide new information about changes in the system and their existence just increases the dimensionality of the problem. Second, different features have different range of values. For example, CPU values can be between 0 and 100, but memory can vary from 0 to 8192. Therefore, we apply a standard normalization to convert all the values to a range between 0 to 1.

Another point to mention is that collected datasets include values from different features over a period of time that create a time series of each feature. Existing trends and seasonality characteristics related to these time series change the pattern of normal data over time. Therefore, as Algorithm 1 shows, we have used STL (Seasonal and Trend decomposition using Loess) technique [11] which is a filtering procedure based on LOESS (local polynomial regression fitting) smoothing to decompose series of various features and extract the trend and seasonality components. However, as [12] shows, the trend component obtained with this method has a problem of introducing some artificial anomalies in the remainder of data which consequently affects the accuracy of anomaly detection algorithms. Therefore, we obtain the Piecewise Median introduced by [12] to calculate approximate trend of time series. The median has shown to be a more robust metric in the presence of anomalies compared to the average values. Having the seasonality and trend components, the remainder component of time series is calculated and have been used as extra features for each dataset.

3.3. Feature Smoothing and Time Dependent Information

The collected datasets have similar characteristics to time series that present the patterns in data through underlying trend and seasonality features. According to our observations, there are some transient spikes

Algorithm 1: Data Preprocessing

input : $D = (X_1, X_2, ..., X_m)$, $X_i \in R^{n\times 1}$: D is a matrix of n records, each record including m features

Parameter: k: Moving Average Window Size

w: Piecewise Median Window Size

output : s: Normalized Extended Data

$s \leftarrow \emptyset$

for $X \in D$ **do**

  Extract seasonal S_x component using STL method from X

  Extract trend T_x component using Piecewise Median from X

  $R_x \leftarrow X - S_x - T_x$

  $s \leftarrow s \cup R_x$

end

for $X \in D$ **do**

  Normalize X

  `// Compute K-moving average`

  initialize all indicators in a_j to 0

  for $j \leftarrow 1$ **to** n **do**

    $a[j] \leftarrow Average(X[max(1, j-k) : max(1, j-1)])$

  end

  $s \leftarrow s \cup a$

end

return (s)

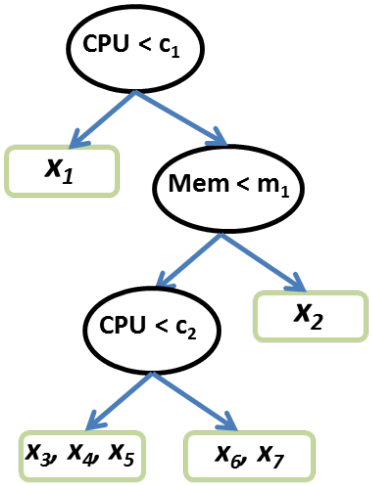


Figure 3: A simple Isolation Tree for two attributes CPU and Memory.

and noises which are introduced during runs of the benchmark. The noise can be as result of wrong or missing measurements or caused by specific characteristics of the benchmark at the start of each run. In order to reduce the effects of transient values, one can consider an average of the consecutive values in predefined time windows which help to smooth these variations. On the other hand, the time of occurrence for each measurement is an important feature which can affect the interpretation of the state of the system. In other words, the behavior of the system presented by nearby instances can affect the decisions to identify an instance as an anomaly or not. The average value of the recently observed instances can help to have an understanding of the previous state of the system and better highlight significant changes between adjacent time windows. Therefore, to further improve our model and include the basic knowledge of time dependent changes in the dataset, we can extend the raw data with a summary of historical data as presented in lines 10-14 of Algorithm 1. To achieve this goal, a window of k previous samples for each instance is considered and the values of the attributes are averaged out, including them as new features for each sample. The technique which is known as k-point moving average, helps to decrease the effect of transient spikes and adds time dependent information into the attribute space.

3.4. Anomaly Detection Approach

Upon receiving enough observations from data preparation module, IFAD is able to start the anomaly detection process. This process can be divided into two main parts: model generation based on the training data and anomaly identification for the test data. In the following, we explain these two parts in more details.

IFAD leverages Isolation technique as the core part of its functionality. This technique is a decision-tree based ensemble approach named Isolation Forest (IForest) introduced in [13, 5]. We choose Isolation technique for our anomaly detection problem based on our observations that the target types of anomaly in the cloud performance data usually change the values of metrics suddenly and these changes are rare

compared to the normal behavior of the system. Therefore, we suggest that there is a high chance that these rare unseen values can be detected based on the partitioning of attribute space in the presence of normal points. This will eliminate the need for calculating distance or density which results in high memory and computation complexities. Traditional classification methods cannot deal with highly skewed distributions. Anomaly detection is a classic example of highly skewed data. Isolation based technique is extremely fast and have been shown to work with a wide variety of distributions of data and do not require prior knowledge of these distributions. They can also be run in parallel to detect anomalies and its cost is negligible compared to many existing anomaly detection algorithms.

The basic assumption of IForest is that anomalies are rare and different and as a result, anomaly instances can be isolated faster than normal ones in the attribute space. This problem is formulated as a binary tree and each node is created by blindly (no need for the labelled dataset) selecting features and values as the conditions to split existing instances. The input of IForest for the model generation part is a sequence of n observations with extended attributes prepared as described in sections 3.1 and 3.2 to be used as the training data. In order to generate the first binary tree, IForest randomly selects $\psi <= n$ instances from the input observations. An attribute c from the column space of the training data and a value for the attribute is selected. Then, all ψ instances are divided into two categories based on the comparison of their values for the attribute c. The generated categories are assigned to two new nodes that create left and right children for the root node of the tree. The process of selecting an attribute and dividing existing instances into sub nodes repeats for new children nodes until the termination conditions are met, which means that there is only one instance left at the node or all the instances have the same values or the maximum length for the tree has been reached. Figure 3 shows a simple Isolation Tree for two attributes CPU and memory. Let $X = (x_1, x_2, ..., x_7)$ be a subsample of input observations to be isolated by their two columns. The root node divides X by selecting value c_1 for attribute CPU. As you can see in the figure, all instances except x_1 are moved to the right child node and x_1, as the only instance left, creates a leaf node at the left of the tree. The right child node divides remaining instances by selecting value m_1 for memory which creates a leaf node at the right containing x_2 instance while other instances move to the left child node. This process continues until the conditions for termination are met. As we can see, instance x_1 can be a possible anomaly point in our sample set as it seems to be in a different range of CPU values compared to the other instances. To create an ensemble of the binary trees, this process repeats to generate t trees. The ensemble represents an abstract model of the current state of the system that can be referenced to find behavior deviations from the past.

The second part of the problem is the identification of the anomalies in the test data. Every test observation should traverse all the generated trees based on its values for the selected attribute of each node until it reaches to a leaf node. The path length of the tree from the root to the leaf node represents the number of required partitions to isolate an instance on its values. The anomaly score is calculated based on the average path length of traversing the trees using a formula presented by [5].

We should highlight two points regarding the adaptability of Isolation technique in our target performance analysis problem. First, this method is an unsupervised learning approach which shows a *low linear-time complexity* with *small memory requirements. These are essential characteristics for the problem of performance management in clouds to have fast and low-overhead solutions with the capability of finding previously unseen performance problems. Indeed, the worst time for training and testing of the algorithm is* $O(t\psi^2)$ *and* $O(mt\psi)$ *respectively, where* ψ *is the number of selected subsamples and* m *is the size of testing dataset. This also leads to the conclusion that the training complexity is constant when the subsample size and the number of trees in the ensemble are fixed [5]. Furthermore, for the problem of anomaly detection in highly dynamic environments, there is a significant issue that usually is neglected about the impact of the workload heterogeneity in the accuracy of the models. The heterogeneity in web applications due to the resource configurations or internal and external events can change the normal pattern of data in the system. A fast anomaly detection procedure which does not require time consuming parameter tunings is an essential requirement which is satisfied by the IFAD framework.*

3.5. Evaluation Metrics

In order to evaluate the detection accuracy of different algorithms, we distinguish four cases. True Positive (TP) that represents a case that an anomaly instance is reported correctly by the algorithm as anomaly. False Negative (FN) that shows a missed anomaly instance which is not detected by our algorithm. False Positive (FP) which refers to cases that normal instances are detected as anomaly and True Negative (TN) that shows a normal instance is correctly identified as normal. Considering these definitions, two metrics, True Positive Rate (TPR) and False Positive Rate (FPR) can be calculated based on Eq. 1 and Eq. 2. In probability based detection algorithms that calculate anomaly scores for each instance, the values TPR and FPR depend on the selection of a threshold that distinguishes anomaly instances from normal ones. Receiver Operating Characteristics (ROC) is a curve that represents a trade-off between TPR and FPR for different thresholds (cutoffs) of anomaly scores. Most of the existing works report the Area Under the Curve (AUC) for this curve as a measure of detection capability of an algorithm for target datasets.

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

However, in the addressed problem to identify anomalies in the cloud environment, the number of normal instances are much larger than anomaly instances which means that the positive and negative class labels are highly unbalanced. In the cases with highly unbalanced values for class labels, [14] shows that the RPAUC value may capture some patterns in the detection efficiency of algorithms that cannot be represented by ROC curve. PRAUC is calculated as the area under the curve for Precision and Recall values which can be calculated based on Eq. 3 and Eq. 4. Precision is a measure of the fraction of detected anomalies that

are true anomalies and Recall is a measure of the fraction of true anomalies that can be detected by the algorithm.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

These metrics combined together, enable us to have a better understanding of real capabilities of different algorithms for evaluation.

4. Performance Evaluation

The experiments are performed by deploying a realistic web serving benchmark on the Australian Nectar cloud environment. In order to select a proper benchmark, some of the existing benchmarks such as RUBIS and PetStore that are extensively used in the literature to monitor the performance of VMs were considered [9, 15] . However, these benchmarks cannot capture the interactive functionalities of today's Web 2.0 applications. Therefore, for the implementation part, a 3-tier web application based on CloudStone benchmark is deployed [10]. CloudStone is part of the CloudSuite which is a benchmark for cloud services. The benchmark highlights the distinctive characteristics of Web 2.0 workloads compared to previously mentioned benchmarks and aims to generate real web workloads to capture web functionality in a scalable cloud environment. The 3 main layers of the benchmark are shown in Figure 4. It includes a Markov-based workload generator for emulating user requests, application and database servers. Workload generator enables the benchmark to have a fine-grained control over parameters that characterize the workload behavior [10]. CloudStone employs Faban, deployed on all the machines, to control the runs and emulate the user behavior. Application servers host a PHP based social network application in nginx servers. The generated requests sent from Faban client, are processed in application and database servers and the results are sent back to the client machine.

Benchmark represents a Web 2.0 social event application that mimics real user behavior in an interactive social environment with a combination of individual and social operations (such as creating events, tagging, attending an event or adding comments). Each request from the user includes a sequence of HTTP interactions between client and server which accomplishes one of the mentioned tasks. We have also installed Ganglia that is a scalable, distributed monitoring component to monitor and collect performance indicators of system and applications.

4.1. Data Generation and Anomaly Injection

For the purpose of evaluating the performance of IFAD on workloads with different characteristics, five datasets are collected based on the specifications defined in Section 3.1 and each dataset includes about

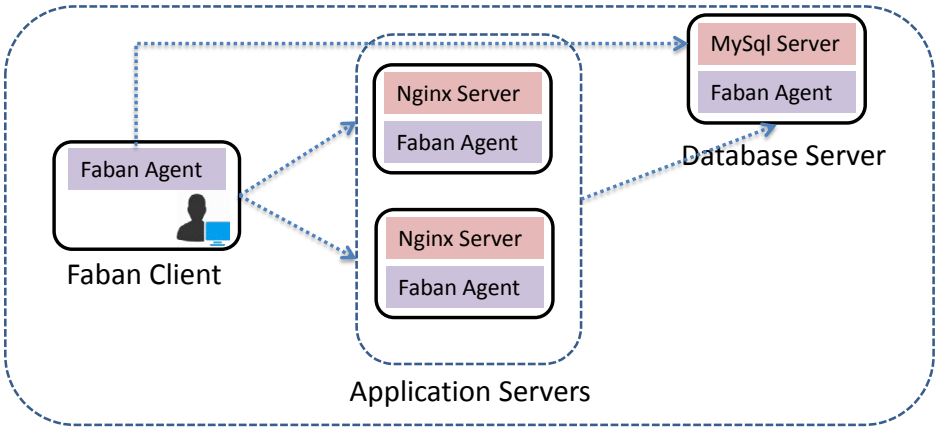


Figure 4: CloudStone Components.

15 hours of performance data. In order to generate each dataset, the workload generator starts to send a sequence of requests to the web server as part of the normal behavior of the system which generates time-series of performance and utilization data resulted from the interactions of user with the application. Then, anomalies are injected at random times to VMs hosting application or database server. The duration of different types of the anomaly may differ, but the contamination rate of the final data with anomaly instances is kept in the range 7-11% for all experiments. This is a rate that corresponds to a low anomaly intensity which is more common in cloud environment [6]. In the following experiments, five types of anomalies are tested:

Memory Load: A process is started on the same VM hosting the application server that allocates the available memory of the VM, but forgets to release it. As a result, after some time, the web application server encounters the problem of finding the required memory to process requests received as part of the normal operation of the system.

CPU Load: A CPU-intensive process is started on the VM that hosts the application server.

Disk Load: A series of I/O intensive tasks (read and write multiple files) are performed on the VM that hosts the database server.

Server Fault: The application server is shut down for some time. As there is no server available to respond to the requests, the utilization of the host VM decreases without any significant change in the number of incoming requests.

Flash Crowd: The number of concurrent users is suddenly increased to simulate a spike in the number of requests. Therefore, all measurements show a higher utilization of VM resources in response to this change.

The anomaly injection scripts are generated with the help of a variety of the packages including stress-ng and Cpulimit and the generator files for different types of anomalies are distributed to the target machines to be called by the master node at identified start times.

The final dataset, after applying preparation filters and adding new features, includes 29 features for the

Table 3: Experiment Configurations

Variable	Description	Value
Anomaly Contamination	The rate of anomaly instances in each dataset	$7\% - 11\%$
t	Number of trees	100
ψ	Number of random samples selected from each dataset	256
n	Total number of instances in training dataset	1650 - 1850

application server and 69 features for the database server. The dataset includes various types of performance indicators such as CPU and memory. Table 1 shows some of the major attributes collected for this dataset.

4.2. IFAD Settings

The base functionality of IFAD is on the assumption that anomalies are rare and different. To achieve this goal, IForest builds an ensemble of trees on a selected sample of data. The anomaly points are detected as instances with the shortest average path length on the generated trees. Corresponding to each node of the tree, one attribute is selected and the existing instances at the node are partitioned, creating two nodes based on the values of the selected attribute. In this work, we apply two different approaches for attribute selection phase of this algorithm:

- Random IForest (**IForestR**): This is the default procedure for attribute selection which splits each node of the tree based on a randomly selected attribute and a random value for this attribute.
- Deterministic IForest (**IForestD**) which tries to select an attribute that best divides the sample space into two categories with different distributions.

We develop and test anomaly detection models in IFAD using IsolationForest package implemented in R environment.[3] The training parameters used in all the evaluations are the same and equal to the values presented in Table 3.

4.3. Evaluation Results

To validate system anomaly detection for web applications, we have conducted extensive experiments to evaluate different aspects of IFAD using several datasets collected from the deployed environment on Nectar

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

Table 4: AUC of all methods

	IForestD	KNN	OCSVM	L2SH	IForestR
Dataset1	**90.3**	**95.0**	**95.2**	**94.8**	**92.4**
Dataset2	79.0	**83.2**	80.3	78.1	**87.0**
Dataset3	**92.2**	**92.0**	**91.3**	**91.5**	**88.9**
Dataset4	**88.3**	71.9	65.9	68.8	73.8
Dataset5	86.1	**92.0**	86.1	**88.5**	**89.6**

virtualised environment. To perform comparisons, three unsupervised algorithms are also implemented as follows:

- KNN: The k-nearest neighbor distance is computed for each sample as a score of anomaly. The curve is computed based on the adjustment of cutoff value on the distance measure. In order to select k, we have tested different values from 2 to 10 and based on the results, a proper value is selected.
- One-class SVM (OCSVM): OCSVM is another algorithm with a non-linear kernel which calculates a soft boundary of normal instances and identifies outliers as data points that do not belong to the normal set. OCSVM is basically used for the novelty detection. However, as the selection of decision boundaries are soft, it can be applied in unsupervised problems as well. In order to select kernel parameter, we have tested different configurations through a 5-fold cross validation and selected the parameter γ based on the best results.
- L2SH: L2SH is a family of Locality-Sensitive Hashing isolation forests (LSHiForest) proposed by [16]. LSHiForest is a generalized version of the isolation based anomaly detection forests in which IForest and L2SH are two special cases of this family applying different types of the similarity measure and LSH functions. The base idea of LSH functions is that similar instances should be hashed to the same bucket with a higher probability than other non-similar instances. Therefore, in LSH trees, an internal node can be partitioned into more than two branches which is dependent on the number of buckets from hashing procedure. Regarding similarity measures, L2SH is associated with l2-norm or Euclidean distance.

Algorithms with random characteristics including IForestD, IForestR and L2SH are repeated 10 times

Table 5: PRAUC of all methods

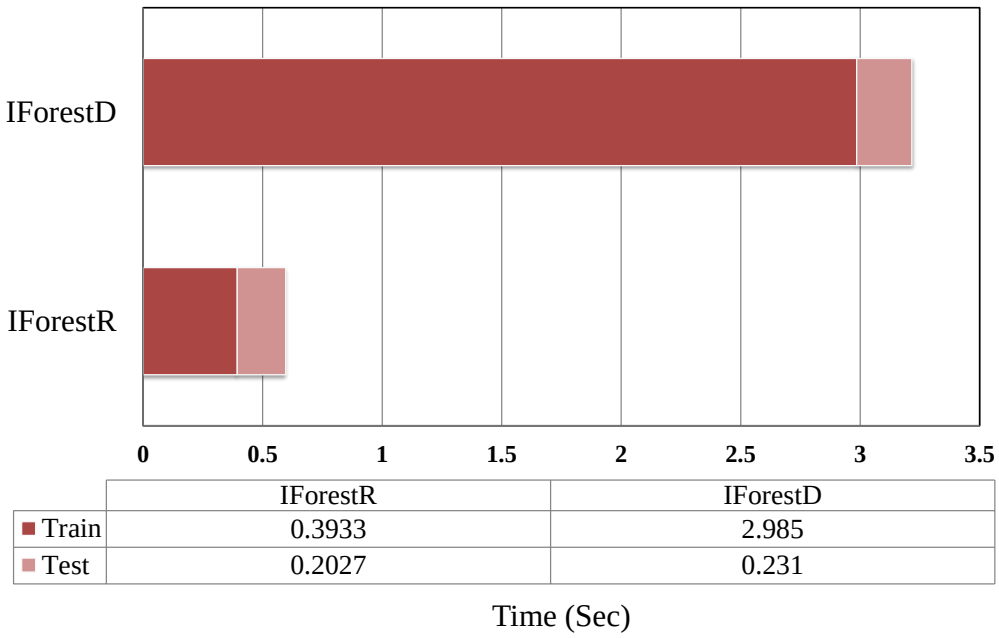
	IForestD	KNN	OCSVM	L2SH	IForestR
Dataset1	**75.9**	57.0	68.8	66.3	54.5
Dataset2	**47.0**	**44.0**	36.8	36.9	**45.1**
Dataset3	**67.6**	39.5	44.8	43.8	40.8
Dataset4	**54.7**	44.4	35.8	35.8	40.7
Dataset5	**50.0**	**53.6**	**47.3**	**50.8**	**49.2**

in each experiment and the average of the results are reported. Though our methods are unsupervised, to be able to validate the accuracy of algorithms, we track the time of the anomaly injection and consider the indices of corresponding measurements as the true anomaly points and the remaining measurements as true normal points.

For the first scenario, we train algorithms with both normal and abnormal instances. Then, each model is tested on another part of the data that includes all types of anomalies. The results for both metrics AUC and PRAUC are reported for all the datasets in Table 4 and Table 5. For each dataset, the best results with a difference of maximum 5 percent are highlighted. As the result shows, IForest can detect anomalies with high accuracy and performs particularly well in PRAUC with the highest results in all datasets.

KNN and IForestR perform well in 4 out of 5 datasets followed by IForesD and L2SH in terms of the AUC while IForestD also achieves high PRAUC for all the datasets. These observations are expected as IForestD tries to select the best splitting attribute at each node so there is a higher probability to isolate anomaly points with a very different distribution than normal points at the top of the trees while it may miss some points with more similar distributions to the normal space. For dataset1 and dataset3 other algorithms also show good AUC while their PRAUC is less than IForestD. Regarding dataset2 which shows a high variance in the values of collected attributes, anomaly instances can hardly be detected and as a result, the average precisions of all algorithms are low. Regarding other datasets, as the variance of data increases, data becomes more scattered and the pattern of anomalies can be masked by some of the normal instances resulting from high fluctuations in the data. In all these cases, IForestD shows good AUC and PRAUC compared to other algorithms.

Figure 5 shows a comparison of average training and testing times on all datasets between two versions



	IForestR	IForestD
Train	0.3933	2.985
Test	0.2027	0.231

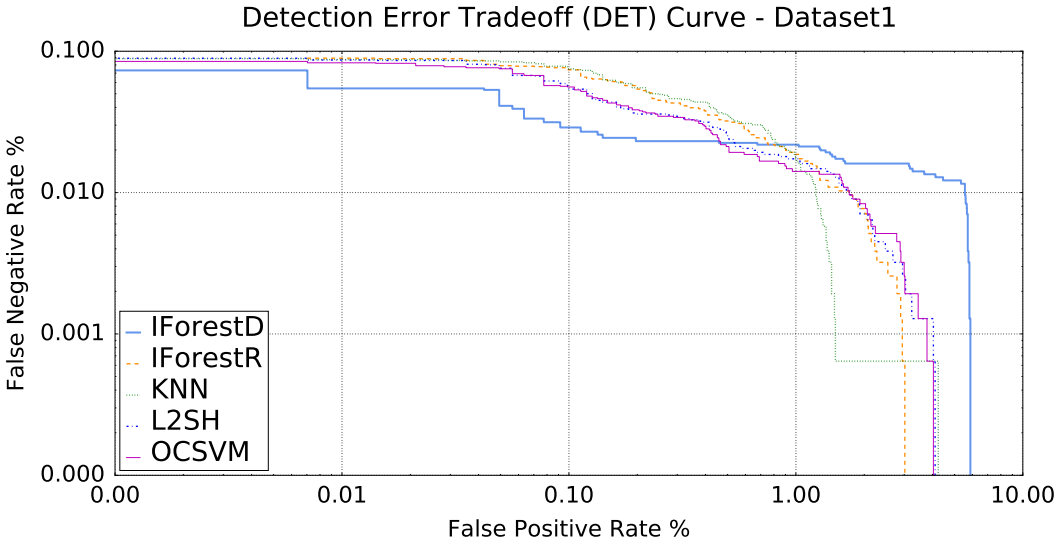
Figure 5: A comparison of train and test times for IForestR and IForestD. The average testing time for one instance is around 0.1 milliseconds considering the size of test datasets for different workloads.

of IForest used in this paper to better demonstrate the effect of attribute selection complexity on the timing of the IForest. We observe that IForestR which selects the attributes randomly is very fast with a training time less than 1 second while IForestD has a training time around 3 seconds which is slower than the random version. However, in the worst case, updating of the models happens at each monitoring interval which is 15 seconds in our work and this interval can be higher based on the stability of application and environment [6]. Moreover, considering the number of instances in the test data, testing for one instance takes around 0.1 milliseconds. This is a reasonable result especially considering that the booting of new VM instances can take around 2 minutes or more based on the performance study done by [17].

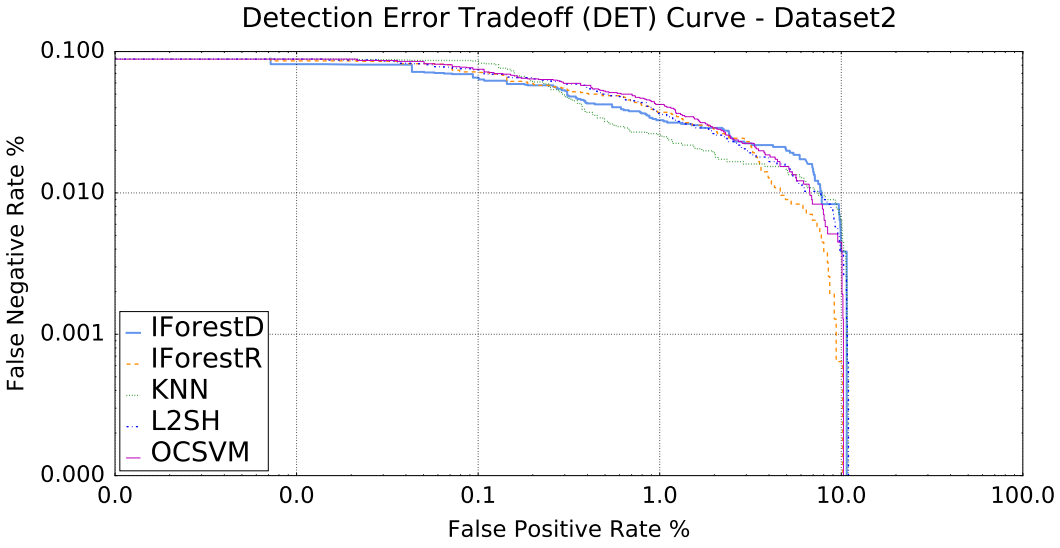
Figure 6 depicts the detection error trade-off (DET) curves for all algorithms and for each dataset. The curves are computed based on defining different thresholds on anomaly scores and computing the log rate of the missed anomalies (FN) and false alerts (FP). FN represents the rate of missing anomaly cases while FP is a measure of false alarms that can wrongly cause the application administrators to start preventive actions. This trade-off is an important observation especially for the applications that have tight restrictions on the accepted rate of positive/negative false alerts [3]. As we can see, no algorithm shows the best FP and FN for all thresholds or datasets which is expected especially due to the heterogeneity of datasets. For example, IForestD performs better in dataset1, dataset2 or dataset3 for FP less than 1%. The observed results confirm the idea that we need to have a more precise understanding of the real requirements of application to be able to select proper approaches that fit the specifications of our problem. This can be achieved by manually identifying the preference of the applications in terms of the precision or recall values or using the concepts of majority voting and ensemble approaches which try to combine the results of several algorithms. As an example case, for prevention mechanisms that target disk related problems with expensive mitigation actions, one may prefer to have a method that has a high precision with minimum false alarms. In contrast, for the Load problems, the high detection rate of the problem may be more important, so an algorithm with a better recall value is preferred.

For the second set of experiments, we investigate the detection performance of different methods for each type of anomaly. The results are shown in Table 6 and Table 7. All methods have a high AUC value for CPU anomaly which shows they can accurately identify anomaly points corresponding to the high utilization of CPU. However, IForestD also shows a high precision for this type of anomaly that represents a lower false alarm rate. For Disk and Server anomalies, IForesetD again shows a high AUC and RPAUC value compared to other algorithms. However, it has a low precision for memory and Load anomalies. The reason can be due to the gradual increase of these two types of anomalies that create a denser cluster of anomaly points which can decrease the difference between normal and abnormal anomaly scores. L2SH has a more stable performance in these cases and usually avoids the worst case performance in different scenarios.

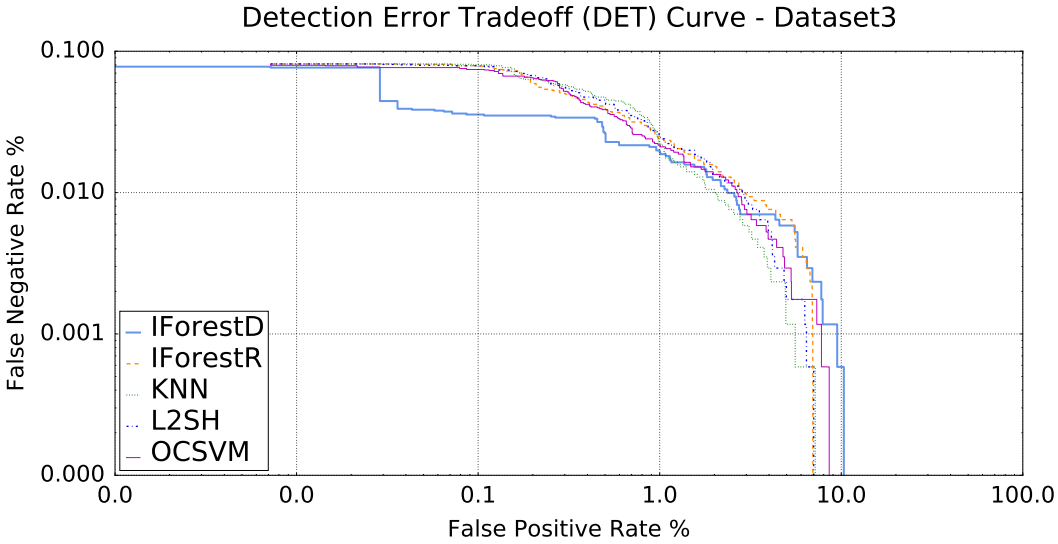
Finally, we show the effect of multi-attribute compared to the single attribute performance analysis. We have repeated experiments with the IForestD algorithm for three scenarios of feature selection. In the first run, we include the CPU metric as the only feature to detect anomalies. In the second run, we add the



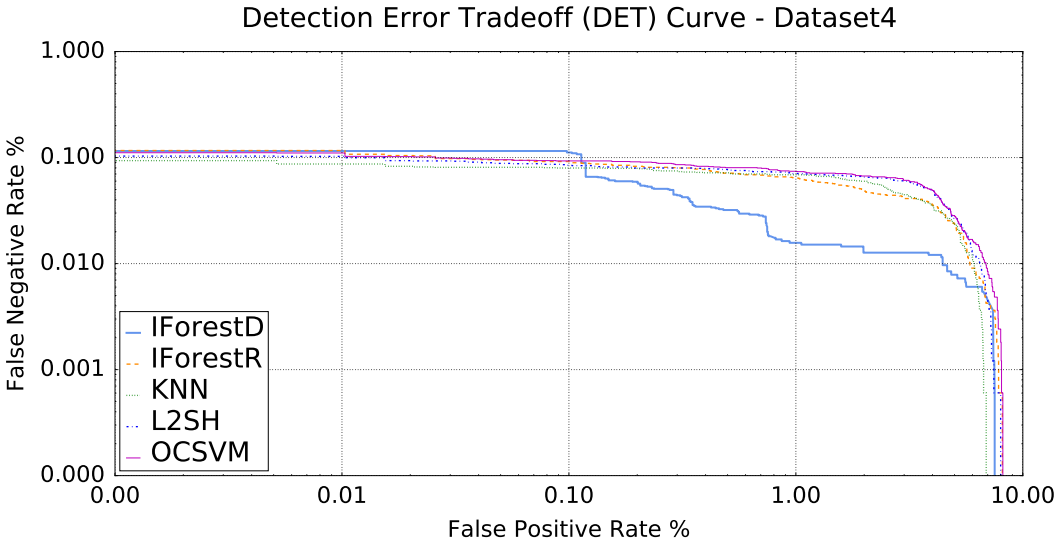
(a) DET Curves - Dataset1



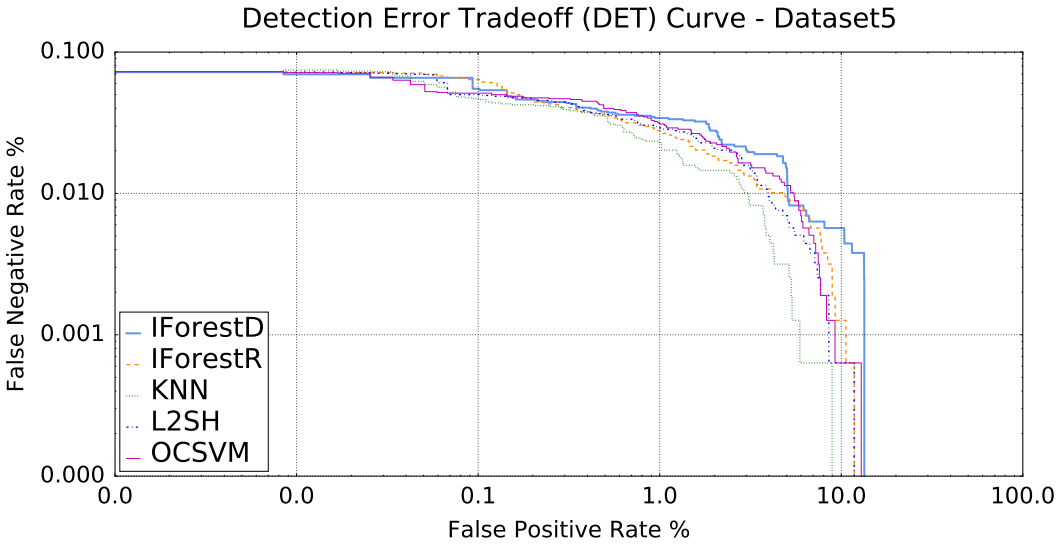
(b) DET Curves - Dataset2



(c) DET Curves - Dataset3



(d) DET Curves - Dataset4

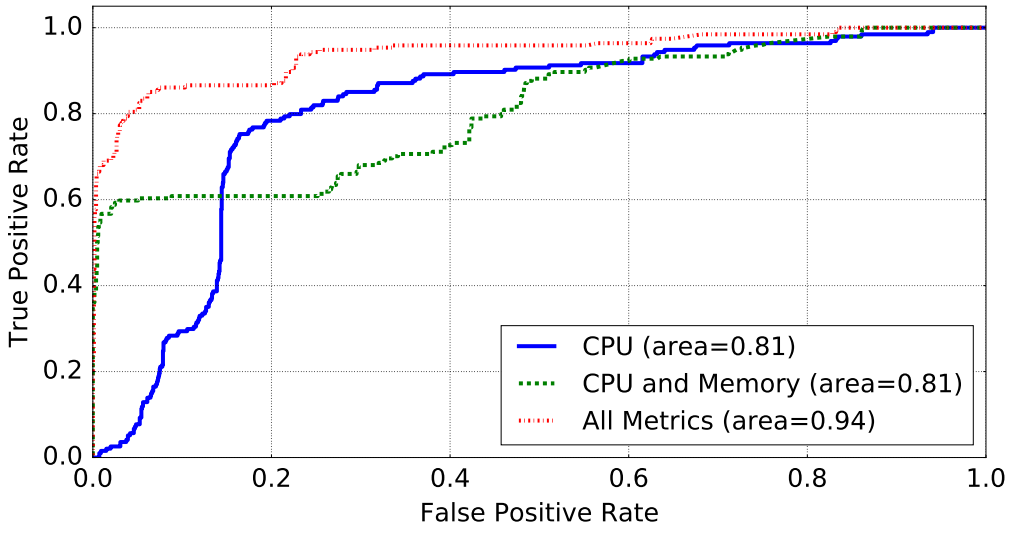


(e) DET Curves - Dataset5

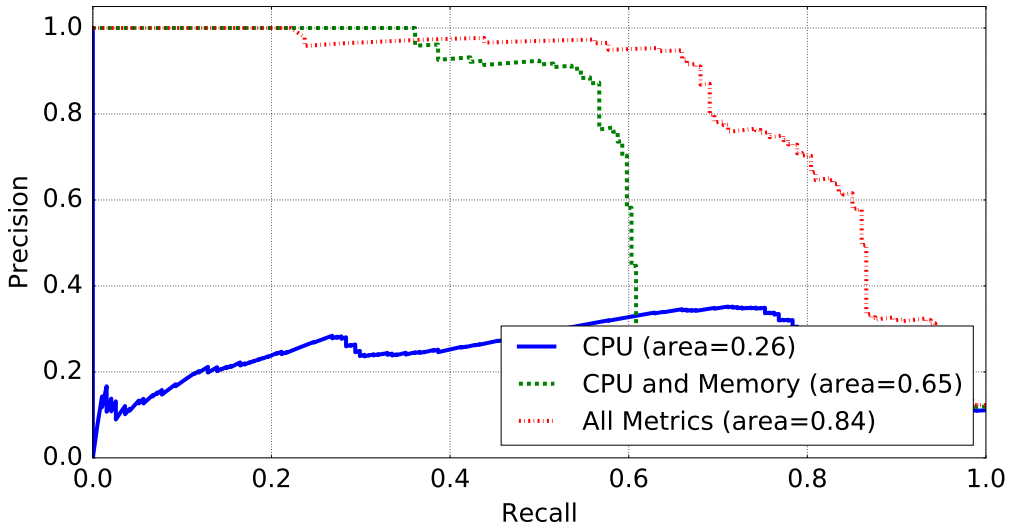
Figure 6: Plots of Detection Error Trade-off (DET) curves for all algorithms and different datasets

Table 6: Anomaly Detection for each type - AUC of all methods

	IForestD	KNN	OCSVM	L2SH	IForestR
Memory	88.2	**94.0**	**93.0**	82.4	75.8
Disk	**97.5**	90.7	**95.9**	**96.4**	**96.1**
CPU	**99.4**	**96.0**	**96.7**	**98.44**	90.4
Load	88.2	**96.8**	**99.8**	**99.2**	**99.9**
Server	**96.4**	90.2	**92.3**	**93.4**	**93.9**



(a) ROC Curves



(b) RPROC Curves

Figure 7: Plots of ROC and PRROC for IForestD algorithm based on different metrics

Table 7: Anomaly Detection for each type - PRAUC of all methods

	IForestD	KNN	OCSVM	L2SH	IForestR
Memory	44.1	68.9	**89.2**	64.8	50.9
Disk	**95.1**	32.8	67.0	84.0	78.8
CPU	**93.9**	75.4	79.7	86.0	59.5
Load	44.1	68.6	**98.7**	91.0	**99.5**
Server	**76.4**	46.1	58.3	67.2	69.4

Memory feature and obtain the result of anomaly detection based on a combination of two features. We compare these two scenarios with another run of the algorithm on all collected features. The comparison is performed by measuring AUC and PRAUC metrics and shown in Figure 7. We can see that the single metric of CPU is not that much informative as we miss many anomaly points and the precision of detection is very low. When we consider both features of CPU and Memory, the results of AUC and PRAUC show significant improvements. However, including all the metrics shows further improvements in the results of the anomaly detection. This leads us to conclude that in a dynamic environment with different types of anomalous problems, a combination of multiple metrics is much more informative and precise than single-feature based solutions.

In summary, the proposed IFAD framework shows higher levels of precision for a range of datasets and anomalies. These results accompanying the unsupervised fast execution of anomaly detection process and the ability to work with the default configurations in various types of workloads which reduces the overhead of tuning steps during model updating makes it a good candidate for applications with a highly dynamic nature, demanding higher precisions or requiring to perform in completely unsupervised manner.

4.4. Time Complexity

In order to have a better understanding of the performance of proposed method, we identify the main blocks of preprocessing and behavior learning steps as follows: The main parts of any anomaly detection framework are data preparation and model generation/testing. Algorithms 1 shows the detailed steps of data preparation based on the concepts of time series analysis. The input is a matrix of n rows (instances) with m columns (features). Assuming fixed seasonality patterns with default parameters, the complexity of data preparation step which is dominated by STL process is O(mn).

Considering that all target models in our work can take the advantage of detrending and seasonality smoothing done in the preparation phase, the main difference in the runtime complexity of the evaluated learning algorithms comes from model generation and parameter tuning. As explained in [5] and section 3.4, considering a constant number of trees and sub sampling size for each isolation tree, the training and space complexities of IForest are constant which makes it suitable for large datasets. L2SH is another version of Isolation-Tree based methods that utilizes locality sensitive hashing. While the distance measure is different compared to IForest version, it shows the same runtime complexity[16]. In contrast, OCSVM and KNN both needs the pre-tuning of parameters and show higher runtime complexities. OCSVM involves a quadratic programming problem which increases the complexity between O(n^2) to O(mn^2) depending on the cashing capabilities and the sparsity of columns. The KNN algorithm requires the computation of the distance to recognize anomaly points. The referenced K for distance calculations highly depends on the distribution of data and one needs a careful testing of different K values especially when the workload characteristics change over time. Having an efficient data structure for implementation, the complexity of the algorithm can be improved to O($mlogn$). Referring to the comprehensive analysis of Isolation-Tree based methods in [5] which shows the robustness of the algorithm with the default values of parameters (100 trees, 256 sample size) and the possibility of having a parallel implementation for ensemble trees generation to further improve the speed, Isolatin-Tree based anomaly detection shows a promising capability for environments where the models need to be updated regularly.

5. Related work

In this work we have proposed an Isolation based anomaly detection framework to detect performance anomalies in the 3-tier web-based applications and investigated the effectiveness of multiple algorithms based on AUC, PRAUC and DET measurements. This is a starting point to give insights to system administrators about the importance of the specific requirements of their application in selecting a suitable data analysis approach. In this section, we first discuss anomaly detection algorithms in general and then focus on the anomaly detection applications in cloud environment.

5.1. Anomaly Detection

The concept of anomaly detection has been widely studied under different names outlier or novelty detection, finding surprising patterns or fault and bottleneck detection in operational systems. There are a variety of survey and review papers that try to classify existing algorithms based on their requirements and computation approach into different categories[18, 4]. Distance based algorithms utilize an approach that addresses the problem of outlier detection based on the concept of the distance of each instance to the neighborhood objects. Greater the distance of an instance to the surrounding objects, more likely that the instance is an outlier [19]. Another approach defines the local density of target instance as a measure for the degree of outlierness of that instance. Objects that reside in the low degree regions are more likely to

be known as an anomaly [20, 21]. While distance and density based approaches show promising results in various types of the datasets, they usually require complex computations which are not preferable in the high dimensional or fast-changing environments.

Another anomaly detection approach which demonstrates promising characteristics in terms of the time complexity and memory requirements is isolation-based technique [5]. In contrast to the traditional approaches of anomaly detection that anomalies are detected as a by-product of another problem such as classification and clustering, isolation-based technique directly targets the concept of the anomalies based on the idea that an anomaly instance can be isolated quickly in the attribute space of the problem compared to the normal instances. The method in this paper is based on this approach and explained with more details along with time complexity requirements in the section 3.4. This approach has been also explored in other types of the applications such as fraud detection problem. For example, [22] addresses the categorical values and proposes an isolation based anomaly detection based on the horizontal partitioning of the data. They show that the proposed method can detect some of the hidden anomalies in the subsets of data that can be ignored when the whole data is analyzed. However, their method is highly domain specific and needs pre-knowledge of the structure of datasets. Another work by [23] proposes a sequential feature selection and outlier scoring framework which tries to filter the important subset of features. An outlier scoring algorithm calculates the scores and they try to find a regression formula among outlier scores and original features as the predictors. They have also demonstrated their approach based on the isolation technique as outlier scoring algorithm and have shown the effectiveness of proposed filtering approach in the high dimensional data. In contrast, our approach utilizes time series analytics and isolation based technique for detecting bottleneck anomalies in the cloud hosted web applications.

5.2. Anomaly detection in cloud

The idea of using anomaly detection to find faults in the computing and storage systems has been widely investigated. For example, [3] studies specific requirements of disk performance analysis to have a controlled false alarm, proposing improvements on existing algorithms to avoid high penalties during the disk failure analysis. Hence, they propose statistical testing based approaches and multivariate decision rules to predict disk failures with the aim of reducing false alarms in the prediction process. [15] studies the application of tree-augmented Bayesian networks (TAN) classifiers to relate the resource performance metrics to SLO violations for web-based applications. Although they investigate the effect of different workloads and SLO thresholds, their work does not compare TAN performance with other learning algorithms and neither studies the PRAUC or DET metrics as our work. The work presented in [24] investigates the feasibility of isolation technique to detect anomalies in the data from IaaS datacenters. However, their focus is on the behavior of IForest in the presence of seasonality/trends in their dataset and they do not consider types of anomalies or compare the detection capabilities of IForest with other algorithms for different performance problems with a variety of workloads.

[25] addresses the fault localization problem in distributed applications. The proposed framework combines the knowledge of inter-component dependencies and change point selection methods, taking into account that the abnormal changes usually start from the source and propagates to other non-faulty parts based on the interactions of the components. Principal component analysis is another method to analyze the data, especially to reduce the dimensionality of attribute space. Accordingly, [26] presents an automatic anomaly identification technique for adaptively detecting performance anomalies by proposing an idea that a subset of principal components of attributes can be highly correlated to specific failures in the system. In contrast, our work focus on the unsupervised bottleneck anomaly identification and can be used complementary to these works to detect previously unseen anomalies.

[27] addresses the problem of bottleneck and cause diagnosis by finding the correlation among attributes and application performance metrics. A subset of correlated metrics are selected based on the predefined thresholds and are analyzed to find possible causes of performance anomalies which are injected in the simulated data. However, the proposed approach is sensitive to the degree of temporal correlation among attributes. [28] targets the security issues that can arise after migrating VMs to new hosts. They propose a combination of an extended version of Local Outlier Factor (LOF) and Symbolic Aggregate ApproXimation (SAX) to detect and find possible causes of anomalies. The SAX representation helps LOF to consider the time information during analysis. However, LOF is a semi-supervised algorithm which is sensitive to the presence of anomaly in the training data. [29] applies a threshold based approach for the problem of resource management in web applications. The proposed framework starts to add new resources as a response to detected anomalies based on the observed violation of Response-Time or CPU utilization; moreover, a regression-based predictive algorithm method detects over-provisioned resources to be released. The work presented by [8] considers a single attribute, number of required processors at a certain time, for the resource utilization estimation. They propose a combination of machine learning and statistical methods based on the idea that the former is more reliable in long-term prediction whereas the latter can have more accurate predictions for the short-term intervals. However, their prediction does not include the concept of unexpected behaviors resulting from various anomaly sources. Compared to these works, our work is more general in terms of considering richer feature space and other sources of unexpected behavior.

The application of unsupervised Hidden Markov Models to detect cloud performance anomalies is investigated by [6]. They propose a distributed and online anomaly detection framework, focusing on the 3 main attributes of Memory, CPU and disk. Our work, in contrast, targets higher dimensional problems with large number of features and therefore needs faster detection solutions with less computation complexity and adaptation requirements. [30] exploits unsupervised clustering to detect anomaly patterns at the thread and process level. They collect system level metrics based on the application characteristics and utilize DBSCAN method to detect non-normal behaviors. However, their method requires an off-line clustering of the normal data before starting the anomaly detection process.

[31] investigates proactive anomaly detection in data stream processing systems. The target anomalies

are injected and the training phase is done on a labeled dataset of different anomaly occurrences in historical data. [9] addresses the same problem by integrating a 2-dependent Markov model as a predictor and the TAN for anomaly detection. They utilize TAN models to distinguish normal state from the abnormal ones as well as reporting the most related metrics to each type of the anomaly. These works follow a supervised approach, targeting stream processing applications.

In contrast to existing works, our approach investigates the unsupervised anomaly identification in a multi-attribute space for heterogeneous workloads of web applications. Moreover, we highlight the importance of workload characteristics as well as application specific requirements by studying the relation of different measurements obtained from anomaly detection process.

6. Conclusions and Future Work

Auto-scaling frameworks in cloud environment should be aware of the possible problems in a system that can trigger a warning of performance degradations. In order to achieve this goal, we proposed IFAD framework which utilizes the concept of Isolation-Trees to detect abnormal behavior in the time series of performance data collected from the application and underlying resources. In addition, the effects of different performance anomalies on various types of the workload in a web-based environment are investigated. The results show that IFAD achieves good AUC and higher precision in detecting performance anomalies. Another observation highlights that depending on the type of heterogeneity in the workloads or changes in the performance of resources, some algorithms can have a better detection rate or average precision. Moreover, a combination of different metrics can improve the learning process compared to single metric solutions based on the common features CPU or Memory. IFAD can be utilized as the anomaly detection module in a resource auto-scaling framework where the knowledge from detection process can help to recognize the possible anomalies in the system behavior. As our future work, we are using this knowledge to help auto-scalers to trigger corrective actions and reduce the SLA violations by updating the configurations of cloud resources.

Our method addresses the problem of resource bottleneck identification in the web-based application where the target anomalous behavior is due to the large changes of attribute values. However, there is some type of anomalies that can change the patterns of data and therefore may not be detected by simple point by point analysis. This category of the problem requires pattern recognition techniques in the time series data that can find a time period that the behavior of system has changed. A more general framework that can cover both types of point and pattern based anomalies can be investigated for the future work. Moreover, we plan to employ an ensemble of multiple algorithms or hierarchical detection approaches to improve the robustness of anomaly detection. Regarding ensemble methods, one can investigate proper weighting mechanisms for different algorithms based on the workload characteristics and application dependent requirements such as the level of accuracy in the results. In hierarchical approach, we can divide the detection process into two steps, coarse-grained level that focuses on finding any abnormal behavior in the system and a fine-grained

anomaly identification level that aims to distinguish between multiple types of the performance problem.

Acknowledgments: This work is partially supported by a Linkage research project funded by Australian Research Council and CA Technologies. We like to thank Adel Nadjaran Toosi for his comments on improving the paper.

[1] O. Ibidunmoye, F. Hernández-Rodriguez, E. Elmroth, Performance anomaly detection and bottleneck identification, ACM Comput. Surv. 48 (1) (2015) 4:1–4:35.

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

[3] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, C. Elkan, Improved disk-drive failure warnings, IEEE Transactions on Reliability 51 (3) (2002) 350–357.

[4] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM computing surveys (CSUR) 41 (3) (2009) 15.

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

[6] 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.

[7] A. O. Ramirez, Three-tier architecture, Linux J. 2000 (75es).

[8] 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.

[9] 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.

[10] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, D. Patterson, Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0, in: Proceedings of the 1st Workshop on Cloud Computing and Its Applications, Vol. 8, Chicago, Illinois, USA, 2008.

[11] R. B. Cleveland, W. S. Cleveland, J. E. McRae, I. Terpenning, Stl: A seasonal-trend decomposition procedure based on loess, Journal of Official Statistics 6 (1) (1990) 3–73.

[12] O. Vallis, J. Hochenbaum, A. Kejariwal, A novel technique for long-term anomaly detection in the cloud, in: Proceedings of the 6th USENIX Conference on Hot Topics in Cloud Computing, Philadelphia, PA, 2014, pp. 15–15.

[13] F. T. Liu, K. M. Ting, Z.-H. Zhou, Isolation forest, in: Proceedings of the 8th IEEE International Conference on Data Mining, IEEE, 2008, pp. 413–422.

[14] J. Davis, M. Goadrich, The relationship between precision-recall and roc curves, in: Proceedings of the 23rd International Conference on Machine Learning, ICML '06, ACM, New York, NY, USA, 2006, pp. 233–240.

[15] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, J. S. Chase, Correlating instrumentation data to system states: A building block for automated diagnosis and control, in: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, OSDI'04, USENIX Association, 2004, pp. 16–16.

[16] X. Zhang, W. Dou, Q. He, R. Zhou, C. Leckie, R. Kotagiri, Z. Salcic, Lshiforest: A generic framework for fast tree isolation based ensemble anomaly analysis, San Diego, California, USA, 2017, pp. 983–994.

[17] 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, 2012, pp. 423–430.

[18] S. Agrawal, J. Agrawal, Survey on anomaly detection using data mining techniques, Procedia Computer Science 60 (2015) 708–713.

[19] S. Ramaswamy, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 427–438.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, 1996, pp. 226–231.

[21] Y. Zhu, K. M. Ting, M. J. Carman, Density-ratio based clustering for discovering clusters with varying densities, Pattern Recogn. 60 (C) (2016) 983–997.

[22] E. Stripling, B. Baesens, B. Chizi, S. vanden Broucke, Isolation-based conditional anomaly detection on mixed-attribute data to uncover workers' compensation fraud, Decision Support Systems 111 (2018) 13 – 26.

[23] G. Pang, L. Cao, L. Chen, D. Lian, H. Liu, Sparse modeling-based sequential ensemble learning for effective outlier detection in high-dimensional numeric data, Thirty-Second AAAI Conference on Artificial Intelligence. (2018) 3892–3899.

[24] 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 29 (18) (2017) e4169.

[25] H. Nguyen, Z. Shen, Y. Tan, X. Gu, Fchain: Toward black-box online fault localization for cloud systems, in: Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems, IEEE, Philadelphia, PA, USA, 2013, pp. 21–30.

[26] 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.

[27] T. Matsuki, N. Matsuoka, A resource contention analysis framework for diagnosis of application performance anomalies in consolidated cloud environments, in: Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering, ICPE '16, ACM, New York, NY, USA, 2016, pp. 173–184.

[28] T. Huang, Y. Zhu, Y. Wu, S. Bressan, G. Dobbie, Anomaly detection and identification scheme for vm live migration in cloud infrastructure, Future Generation Computer Systems 56 (2016) 736–745.

[29] W. Iqbal, M. N. Dailey, D. Carrera, P. Janecek, Adaptive resource provisioning for read intensive multi-tier applications in the cloud, Future Generation Computer Systems 27 (6) (2011) 871–879.

[30] X. Zhang, F. Meng, P. Chen, J. Xu, Taskinsight: A fine-grained performance anomaly detection and problem locating system, in: Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 917–920.

[31] 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.