Detecting Anomaly in Cloud Platforms Using a Wavelet-Based Framework

David O'Shea[1], Vincent C. Emeakaroha[1](✉), Neil Cafferkey[1], John P. Morrison[1], and Theo Lynn[2]

[1] Irish Centre for Cloud Computing and Commerce, University College Cork, Cork, Ireland
{d.oshea,vc.emeakaroha,n.cafferkey,j.morrison}@cs.ucc.ie

[2] Irish Centre for Cloud Computing and Commerce, Dublin City University, Dublin 9, Ireland
theo.lynn@dcu.ie

Abstract. Cloud computing enables the delivery of compute resources as services in an on-demand fashion. The reliability of these services is of significant importance to their consumers. The presence of anomaly in Cloud platforms can put their reliability into question, since an anomaly indicates deviation from normal behaviour. Monitoring enables efficient Cloud service provisioning management; however, most of the management efforts are focused on the performance of the services and little attention is paid to detecting anomalous behaviour from the gathered monitoring data. In addition, the existing solutions for detecting anomaly in Clouds lacks a multi-dimensional approach. In this chapter, we present a wavelet-based anomaly detection framework that is capable of analysing multiple monitored metrics simultaneously to detect anomalous behaviour. It operates in both frequency and time domains in analysing monitoring data that represents system behaviour. The framework is first trained using over seven days worth of historical monitoring data to identify healthy behaviour. Based on this training, anomalous behaviour can be detected as deviations from the healthy system. The effectiveness of the proposed framework was evaluated based on a Cloud service deployment use-case scenario that produced both healthy and anomalous behaviour.

Keywords: Multi-dimensional anomaly detection · Wavelet transformation · Cloud monitoring · Data analysis · Cloud computing

1 Introduction

Cloud computing has transformed the delivery of IT resources into services that are accessible through the Internet. The large-scale and abstract nature of Cloud platforms is intimidating to both consumers and administrators. This is reflected in the difficulty of managing such systems to provision consistent and reliable Cloud services in order to gain consumer trust and high adoption of Clouds. Anomalous

M. Helfert et al. (Eds.): CLOSER 2016, CCIS 740, pp. 131–150, 2017.
DOI: 10.1007/978-3-319-62594-2_7

behaviour endangers the consistent performance of virtual machines in providing resources for the execution of consumer applications. It is therefore important to have a strategy for detecting anomalous events in Cloud environments to prevent such inconsistency and improve the reliability of Cloud services. To detect anomaly in Clouds, a number of challenges such as the differentiation between normal and anomalous behaviours need to be addressed. In addition, Cloud environments are dynamic. This means that normal behaviour can be continuously unfolding and a current model of normal behaviour may be different in the future.

Moreover, Cloud service provisioning management, based on monitoring, has focused on detecting performance issues and has largely ignored anomaly detection. Most of the existing anomaly detection solutions tend to address a particular fixed formulation of the problem [8,19]. A recent survey on anomaly detection in Clouds [20] has shown the lack of multi-level detection techniques to adequately address reliability issues in Clouds.

In this chapter, we propose a novel anomaly detection framework for detecting anomalies in the behaviour of services hosted on Cloud platforms. The framework consists of a monitoring tool to supervise service execution on Cloud infrastructures, and a wavelet-inspired anomaly detection technique for analysing the monitoring data across Cloud layers and reporting anomalous behaviour. Based on a service-deployment use-case scenario, the detection technique is evaluated to demonstrate its efficiency. The achieved results are compared against existing algorithms to show the technique's significance.

The rest of the chapter is organised as follows: Sect. 2 presents some background knowledge on anomaly detection and discusses categories of anomaly. In Sect. 3, we analyse the related work and differentiate our contributions to it. Section 4 presents the proposed framework, focusing on the monitoring and anomaly detection components, while Sect. 5 describes its implementation details. In Sect. 6, we present the evaluation of the framework and Sect. 7 concludes the chapter.

2 Background

Anomaly detection (or outlier detection) is the identification of items, events or observations that do not conform to an expected pattern or to other items in a data set. In a regular and repeatable time series, a profile of expected behaviour should be easily obtainable. In medicine, one such example is an electrocardiogram (ECG). This is used to classify a patient's heart activity. A medical doctor has been trained to quickly identify anomalous ECGs, or indeed anomalies in an ECG, by studying a large number of healthy ECGs. An analogous health monitor would therefore be desirable in other areas of science, including the health of shared network resources and Cloud-based services [9,14] that are subject to demands that vary greatly and experience periodic growth, seasonal behaviour and random variations. Anomalous behaviour can be the result of unprecedented user requirements, malicious (hacking) activities, or can be symptomatic of issues with the system itself. Before identifying the cause of anomalous behaviour, one must first identify anomalous behaviour by detecting a measurable deviation from the expected behaviour.

2.1 Anomaly Categories

Based on existing research [8], anomalies can be grouped into the following categories:

1. Point Anomaly: This is a situation where an individual data instance can be considered as anomalous with respect to the rest of the data. It is seen as the simplest form of anomaly and most of the existing research on anomaly detection is focused on this category [8].

2. Contextual Anomaly: This represents a data instance that is anomalous in a particular context. It is also known as a conditional anomaly [27]. The context is mostly derived from the structure in the data set and should be included in the problem formulation. Mostly, the choice to use a contextual anomaly detection technique depends on its meaningfulness in the target application domain. For example, where an ambient temperature measurement would be at the lowest during the winter (e.g., $-16\,^{\circ}$C) and peak during the summer (e.g., $38\,^{\circ}$C), a temperature of $80\,^{\circ}$C would be anomalous. However, $80\,^{\circ}$C is an acceptable value in a temperature profile of boiling water. In this category, the availability of contextual attributes is a key factor. In some cases, it is easy to define context and therefore the use of a contextual anomaly detection technique would be appropriate. In some other cases, context definition and the application of such techniques are challenging.

3. Collective Anomaly: This represents a situation where a collection of related data instances is anomalous with respect to the entire data set. The single data instances in this collection may not be anomalous individually but when they occur together, they are considered anomalous. The following sequence of events in a computer network provides an example:

... http-web, **buffer-overflow**, http-web, smtp-mail, **ssh**, smtp-mail, http-web, **ftp**, smtp-mail, http-web ...

The occurrence of the above highlighted events together could signify an attack where the attacker caused a *buffer-overflow* to corrupt the network, and then remotely accessed the machines using SSH and copied data through FTP. In this example, note that each of these events could be normal but their clustering is anomalous.

3 Related Work

Previously, extensive research has been done for anomaly detection in large-scale distributed systems such as Clouds [15,16,20,23,25]. Ibidunmoye *et al.* [20] present a review of the work done in performance anomaly detection and bottleneck identification. They describe the challenges in this area and the extent of the contributed solutions. In addition, they pointed out the lack of multi-level anomaly detection in Clouds. Mi *et al.* [23] present a hierarchical software-orientated approach to anomaly detection in Cloud systems, tracing user requests through VMs (virtual machines), components, modules and finally functions. The authors attempt to identify those software modules that are responsible for system degradation by identifying those that are active during abnormal and normal behaviour of the system. For example, this approach considers a module to be responsible for abnormal system behaviour if its response latency exceeds the required threshold. However, it does not consider a module that finished quickly because of a software crash as contributing to system degradation.

Some current research in anomaly detection relies on fixed thresholds [7,30]. In [30], particular mention is made of the assumption of Gaussian distributed data with defined thresholds. Unfortunately, this places assumptions (and therefore limitations) on the data being analysed if it must fit (or is assumed to fit) a particular distribution. Typically, these thresholds must be calculated completely *a priori* and therefore require anomaly-free time-series data of the system. MASF [7] is one of the more popular threshold-based techniques of use in industry, where thresholds are defined over precise time-intervals (hour-by-hour, day-by-day, etc.). Lin *et al.* [21] firstly use a global locality-preserving projection algorithm for feature extraction, which combines the advantages of PCA (Principal Component Analysis) with LPP (Locality Preserving Projection). They then use an LOF (Local Outlier Factor) based anomaly detection algorithm on the feature data. LOF assigns a factor to each data point that measures how anomalous it is, and considers points whose factor exceeds a certain threshold to be anomalies.

Statistical approaches have also been developed in prior academic work [1,2,5] to extend to multi-dimensional data as well as reducing false positives. However, these methods often require knowledge of the time-series distribution or may not adapt well to an evolving distribution. On the other hand, probabilistic approaches, such as Markov chains [6,26], can produce excellent predictions of a system's behaviour, particularly if the system is periodic with random, memory-less transition between states. The size of the probability matrix will grow with the number of defined Markov states, and this may present an issue when extending to multi-metric analysis. However, all of these methods only rely on time-domain information while more information exists in the frequency domain. Considering the distributed nature of Clouds, it is a prime target for sophisticated intrusion attacks [18] and therefore merits the consideration of all information available.

Recent works [17,22,23,29] have begun to use wavelet transforms (which utilise time and frequency domain information) as part of their pre-processing techniques to identify and characterise anomalies in Cloud-based network systems. Wang *et al.* [29] describe EbAT - an anomaly detection framework that performs real-time wavelet-based analysis to detect and predict anomalies in the behaviour of a utility Cloud. Their system does not require prior knowledge of normal behaviour characteristics, and is scalable to exascale infrastructure. Using the RUBiS benchmark to simulate a typical website, it achieves 57.4% better accuracy than threshold-based methods in detecting uniformly distributed injected anomalies. It aggregates metric data before analysis in order to achieve better scalability. However, it does not consider multi-level anomaly detection. Guan *et al.* [17] present a wavelet-based anomaly detection mechanism that exhibits 93.3% detection sensitivity and a 6.1% false positive rate. The algorithm requires normal runtime Cloud performance training data. However, it is not indicated how transient anomalies in the training data are identified to prevent false negatives. A subset of metrics that optimally characterises anomalies is chosen. Metric space combination is then applied to further reduce the metric space. It is unclear from this approach what would occur if a metric that was initially stable – and therefore excluded from the metrics under consideration – began to exhibit anomalous behaviour. Once an anomaly is detected using this method the metric responsible cannot be identified.

To the best of our knowledge, none of the existing solutions present a wavelet-based multi-level anomaly detection technique that can detect and diagnose the root causes of anomalies across Cloud resource and application layers.

4 Anomaly Detection Framework

This section describes the architecture of our proposed anomaly detection framework. It is designed to address the previously identified challenges. The architecture is capable of handling the service provisioning lifecycle in a Cloud environment, which includes service scheduling, application monitoring, anomaly detection and user notification.

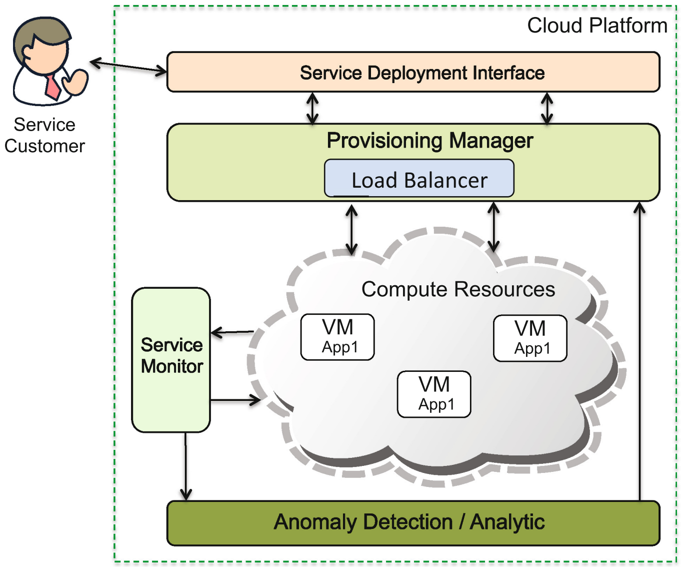


Fig. 1. System architecture.

Figure 1 presents an abstract view of our architecture and its operations. Customers place their service requests through a defined interface (*Service Deployment Interface*), which acts as the front-end in the Cloud environment. The received requests are validated for format correctness before being forwarded to the *Provisioning Manager* for further processing. The provisioning manager includes a *Load Balancer* that is responsible for equally distributing the service/application deployment for optimal performance. The applications are deployed on the *Compute Resources* for execution. The *Service Monitor* supervises the execution of the applications on the compute resources. The resulting monitoring data are forwarded to the *Detection/Analytic* component for analysis. Any anomaly detection from the analysis is communicated to the provisioning manager to take appropriate action.

The proposed architecture is generic to support a wide range of applications, varying from traditional web services to parameter sweep and bag-of-task applications. In this chapter, we detail the monitoring and anomaly detection components.

4.1 Service Monitor Design

The service monitor comprises individual configurable monitoring tools in a decentralised fashion. It is capable of monitoring Cloud resources and applications, which gives it an advantage over resource-monitoring only tools such as *LoM2HiS framework* [10,11]. At the application level, the service monitor supports event-based monitoring of activities. Figure 2 presents an overview of the service monitor.

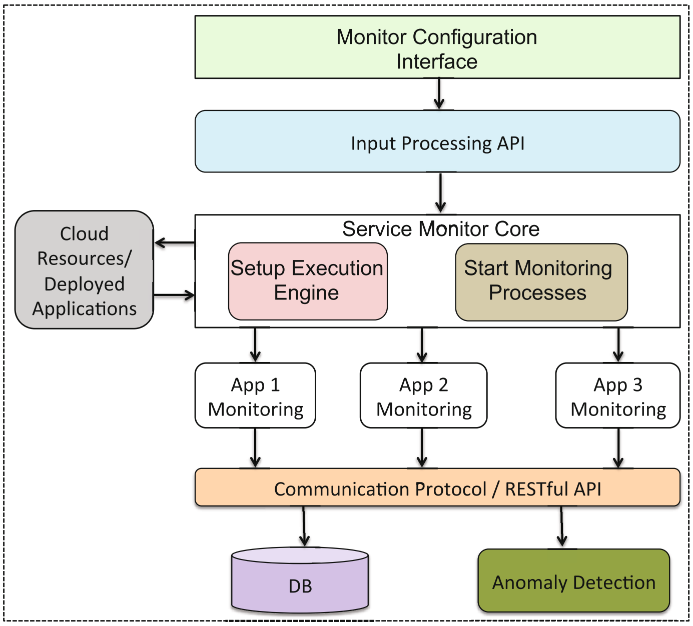

Monitor Configuration Interface
Input Processing API
Cloud Resources/ Deployed Applications
Service Monitor Core
Setup Execution Engine
Start Monitoring Processes
App 1 Monitoring
App 2 Monitoring
App 3 Monitoring
Communication Protocol / RESTful API
DB
Anomaly Detection

Fig. 2. Service monitor.

As shown in Fig. 2, the service monitor has a modular design. The configuration of the tools is done through the *Monitor Configuration Interface*. It allows the parameterisation of the individual monitoring tools, for example to specify different monitoring intervals.

The *Input Processing API* is responsible for gathering the configurations from the previous component and parsing them into a suitable format for the back-end service monitor core engine to understand. The *Service Monitor Core* instantiates the necessary monitoring tools with the proper configuration parameters and supervises them while monitoring the deployed Cloud services. The monitoring tools are executed in parallel and each sends its monitored data using the *Communication Protocol* into a database as well as to the anomaly detection module.

In designing the service monitor, we strived to make it non-intrusive, scalable, interoperable and extensible. These qualities have been associated with efficient monitoring tools as described in a recent monitoring survey [12]. The separation of the service monitor components into modules makes it easily extensible with new functionalities. To achieve non-intrusiveness, we host the monitoring software on separate Cloud nodes to the ones used to execute the customer services. However, we deploy light weight monitoring agents on the compute node for gathering the monitoring data and sending it back to the server. This helps to avoid resource contention between the monitoring server and the deployed Cloud services that might degrade customer service performance. In addition, this separation increases the scalability of the monitoring tool since it facilitates the creation of clusters of monitoring agents with decentralised control servers. The communication protocol uses a platform-neutral data interchange format for formatting and serialising data to achieve interoperability.

4.2 Anomaly Detection Algorithm

Our anomaly detection algorithm is highly configurable for different detection stages. This extended version can tolerate data of an arbitrary rank. In our use case, we consider multiple one-dimensional metrics simultaneously. The wavelet algorithm could also consider matrix and other forms of data streams (e.g. matrix/tensor) if available.

In the algorithm's operation as shown in Fig. 3, we first configure the rank to one (vector data) for N data streams. It currently uses the Morlet wavelet form. Other waveforms can also be used and may be more efficient. An optimal configuration is part of our ongoing work.

The second stage details the training by taking the pre-processed data and performing the wavelet-transform on each metric. The returned spectrograms are then passed to a machine learning technique that has a knowledge of the history of the Cloud system. The newest spectrograms are used to update the running estimate of the mean and standard deviation of an ideal performance. In this way, a profile of behaviour can be extracted, and a deviation from this profile can be identified as an anomaly.

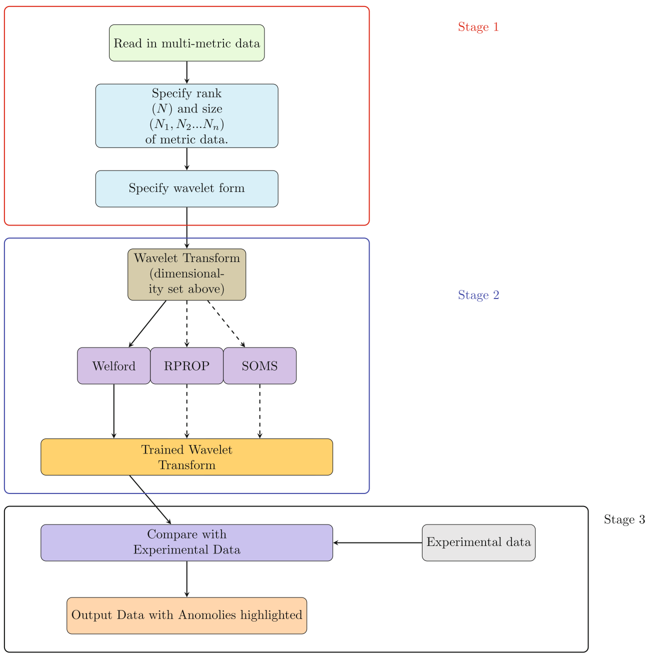

Stage 1
Read in multi-metric data
Specify rank (N) and size ($N_1, N_2 ... N_n$) of metric data.
Specify wavelet form
Stage 2
Wavelet Transform (dimensionality set above)
Welford
RPROP
SOMS
Trained Wavelet Transform
Stage 3
Compare with Experimental Data
Experimental data
Output Data with Anomolies highlighted

Fig. 3. Flow chart of anomaly detection technique.

Thirdly, the trained multi-metric spectrogram is then compared to the multi-metric spectrogram of data under investigation. The steps are outlined in the pseudo code included in Algorithm 1. One benefit of this multi-layer approach is that after having inspected a given data sample for anomalies, the new data can be easily used to extend the usefulness of the trained model and tolerances for normal behaviour can be updated.

The threshold scaling parameter m allows for a specific tolerance to be set for each metric. The wavelet transform is computed using Eq. 1.

$$CWT_x^{\psi}(\tau, s) = \Psi_x^{\psi}(\tau, s) = \frac{1}{\sqrt{|s|}} \int x(t)\psi^*(\frac{t-\tau}{s})dt \tag{1}$$

The mother wavelet (ψ) is a windowing function that scales (s is the scaling parameter and a scalar quantity) and translates (τ is the translation parameter) the time trace ($x(t)$, a vector of metrics recorded simultaneously). A two-dimensional spectrogram (of the complex coefficients) is generated from varying s and τ, for each metric. As s is increased, the time window becomes smaller. This in turn fafects the resolution of frequencies detected in the time traces. The wavelet transform offers superior temporal resolution of the high frequency components and scale (frequency) resolution of the low frequency components. The values of s and τ range from 0 to the length of the time trace undergoing transformation. The exact configuration of the anomaly detection algorithm is introduced in a broad sense so that it can be further optimised without major restructuring.

Algorithm 1. Pseudo code for Multi-metric Wavelet Transform and Welford training.

```
 1: function $Train\_data(timetraces, metric)$
 2:     for $day$ in $timetraces$ do
 3:         $SPEC = wavelet\_transform(rank, N_1, N_2, ....N_n, metric, \Delta t, \Delta\omega, \dots)$
 4:         $M_T, S_T = Welford\_N\_rank(M_T, S_T, SPEC, day)$
        return $M_T, \sqrt{\frac{S_T}{days}}$

 1: function $Check\_for\_anomaly(metrics_A, M_T, S_T, m)$
 2:     $M_A = wavelet\_transform(metrics_A, \Delta t, \Delta\omega, \dots)$
 3:     if $M_A > (M_T + m \cdot |S_T|)$ then
 4:         anomaly found
 5:         Record location to $locs$
 6:     else if $M_A < (M_T - m \cdot |S_T|)$ then
 7:         anomaly found
 8:         Record location to $locs$
 9:     else
10:         no anomalies found
        $Ratio = \frac{M_A}{M_T}$
    return $Ratio$, $locs$

 1: function $Welford\_N\_rank(M, S, SPEC, day))$
 2:     $M_{Temp} = M$
 3:     $M \mathrel{+}= \frac{(SPEC - M)}{day}$
 4:     $S \mathrel{+}= (SPEC - M_{Temp})(SPEC - M)$
    return $M, S$
```

5 Implementation Details

This section describes the implementation of the proposed anomaly detection framework. Our focus is on the monitoring and anomaly detection components.

5.1 Service Monitor Implementation

The monitor configuration interface was realised using Ruby on Rails technology, which enabled rapid development and facilitates its compatibility with other components. A key feature of Ruby on Rails is its support for modularity. We used this feature to make it easily extendible with new functionality. Ruby on Rails also has a rich collection of open source libraries. Based on this, we used the JSON library to aggregate the input configuration data before transferring them down to the next component.

The input processing API component is implemented as a RESTful service in Java. Since Ruby on Rails supports RESTful design, it integrates seamlessly with this component in passing down the input data. The input processing API extracts these data and makes them available to the service monitor core component.

The service monitor core component sets up and manages the execution of user selected and configured monitoring tools. We use multi-threading to achieve parallel execution of the monitoring tools since they are developed as individual applications.

Each monitoring tool incorporates communication protocols for transferring the monitoring data to head components. The communication protocols comprise a messaging bus based on RabbitMQ [28], HTTP and RESTful services. This combination achieves interoperability between platforms. We use a MySQL database to store the monitoring data. Hibernate is used to realise the interaction between the Java classes and the database. With Hibernate, it is easy to exchange database technologies. Thus, the MySQL database could be easily exchanged for another database platform.

5.2 Anomaly Detection Algorithm Implementation

The wavelet transform is implemented in a similar fashion to the wavelet module in MLPY [3]; however, several modifications have been made to accommodate dynamic Cloud metric data. Firstly, the Fast Fourier Transform(FFT) backend is specified using libraries such as scipy, numpy, GSL or FFTW [13]. Secondly, our implementation overcomes dimensionality restriction of data. This means that we are capable of considering a matrix form of data as opposed to the single dimension required by the previous system. We perform a one-dimension, column-wise, multi-scale wavelet transform. That is, the data of an arbitrary N metrics recorded in time are transformed from the time domain to the frequency-time domain across all metrics simultaneously. There exists also an opportunity to use this extended routine further to bring the multi-scale wavelet transform to near real-time applications by executing each individual scale calculation in parallel.

The wavelet transform is usually implemented as part of a larger routine that includes some pre-processing [6,24] and is often trained using an advanced neural network (ANN) such as RPROP (Resilient BackPRoPagation) or SOMS [31]. The routine employed here for the machine learning based on the wavelet transform is outlined in Fig. 3.

In Fig. 3, the solid black arrows indicate the elements of the routine currently available. The dashed black arrows indicate features still in development.

The use of additional ANNs after the Welford Algorithm could allow the identification or correlation of anomalies between metrics.

6 Evaluation

The goal of our evaluation is to demonstrate the efficacy of the proposed framework to monitor Cloud service execution, analyse the monitoring data and detect anomalous behaviour. It is based on a use-case scenario that describes the service interactions. First, we present the evaluation environment and the use-case descriptions.

6.1 Experimental Environment

To set up the experimental environment, an OpenStack Cloud platform installation running Ubuntu Linux was used. The basic hardware and virtual machine configurations of our OpenStack platform are shown in Table 1. We use the Kernel-based Virtual Machine (KVM) hypervisor for hosting the virtual machines.

Table 1. Cloud environment hardware.

Machine Type = Physical Machine				
OS	CPU	Cores	Memory	Storage
OpenStack	Intel Xeon 2.4 GHz	8	12 GB	1 TB
Machine Type = Virtual Machine				
OS	CPU	Cores	Memory	Storage
Linux/Ubuntu	Intel Xeon 2.4 GHz	1	2048 MB	50 GB

As shown in Table 1, the physical machine resources are capable of supporting on-demand starting of multiple virtual machines for hosting different Cloud services.

6.2 Use Case Scenario

This use case scenario describes a Cloud service deployment, the monitoring of the service and the analysis of the monitoring data to detect anomalous behaviour. To realise this, we set up Apache web servers with back-end MySQL databases on our OpenStack platform as the demonstrator Cloud service. On the web servers, we deploy a transactional video-serving web application that responds to requests and makes queries to back-end databases. Video data were uploaded to the web servers that could be rendered on request. The service is designed to receive and process different queries and workloads generated by users.

In this scenario, we simulate user behaviour in terms of generating queries and placing them to the Cloud service using Apache JMeter [4]. The workload consists of three HTTP queries and two video rendering requests. With these queries, we generate approximately 15 requests per second, representing light to moderate load on a real-world service. The video requests invoke playback of music video data on the web servers. We generate five requests per second for two videos in a mixed sequence.

The execution of this service on the web servers was monitored using the service monitor described in Sect. 4.1. The application-level monitor is event-based. Therefore, it can continuously monitor the performance of each request/query placed to the web application. We monitor 74 metrics (such as *BytesReceived, ByteSent, ResponseTime, CPUUserLevel, CPUIdle, FreeDisk, FreeMemory etc.*) from this service deployment.

For this evaluation, we gathered 10 days' worth of data from this service execution monitoring. Since the workload is simulated, the load distribution on each particular machine was repeated each day, therefore the recorded metrics should vary in similar ways each day. No seasonal or periodic effect of the environment on the machines should have occurred; therefore the metric distribution should be normal apart from the presence of small amount of random noise.

Due to the velocity, volume, and real-time nature of Cloud data, it is difficult to obtain time-series data with true labelled anomalies. Moreover, to aptly test our N-metric anomaly detector, a multi-metric anomaly is needed to be present in the monitoring data. To address this, we chose to simulate a Distributed Denial of Service (DDoS) attack by injecting appropriate chosen values into the time trace data from one of the monitored virtual machines (post training). To simulate this attack, a simple anomaly injector was written to inject an anomaly into several metrics including CPUIDLE, CPUUSER, CPUSYSTEM and INBYTES. The DDoS attack was ramped up in 40 s (one time step between measurements), had a duration of around 100 min and then returned to a typical behaviour for the system. Figures 4 and 5 demonstrate this visually.

Given that there is a redundancy between CPUUSER, CPUIDLE and CPUSYSTEM, we will leave out CPUIDLE in this evaluation. We compare the trained spectrograms against the data injected with anomalies in the following sections.

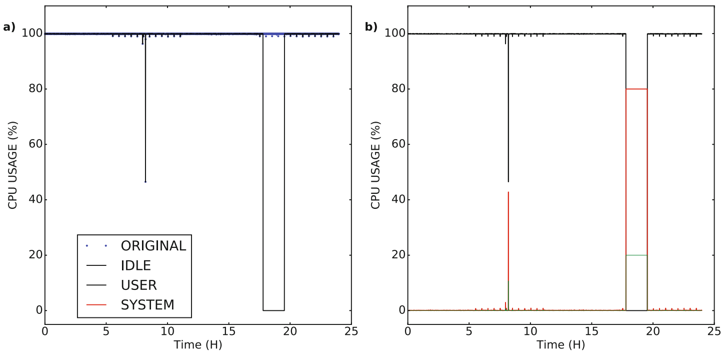


Fig. 4. Sample anomaly injection: **(a)** compares a sample trace (blue) of CPUIDLE against an average time trace with an anomaly injected (black). Here the CPUIDLE drops to zero in under 40 s (one time step between measurements). **(b)** Demonstrates the conservation of total CPU usage between CPUIDLE, CPUUSER and CPUSYSTEM (their sum equals 100%). Here a ratio of 0.8 is used to divide the CPUUSER and CPUSYSTEM levels. (Color figure online)

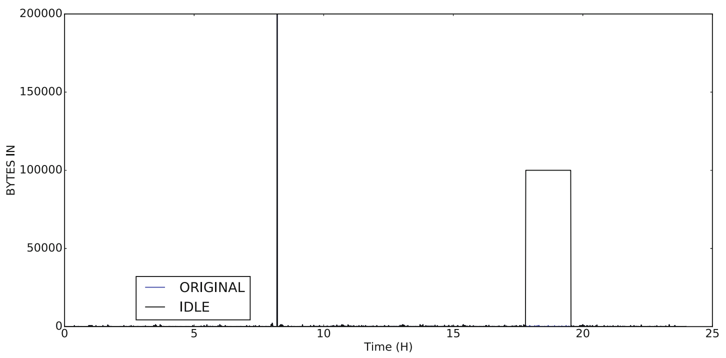


Fig. 5. Sample anomaly injection of INBYTES comparing a sample trace (blue) of INBYTES against an average time trace with an anomaly injected (black). Here the INBYTES increases rapidly over 40 s and maintains a large height for 100 min before dropping back to typical levels. (Color figure online)

6.3 Data Analysis and Results

This section presents the achieved results of our multi-metric anomaly detection framework.

Based on the wavelet algorithm, we generate a separate spectrogram for each day of data. Given that the system load is approximately similar from one day to the next, a typical presence (or absence) of frequency-time events can be detected through the comparison of the individual spectrograms.

To determine if a point is anomalous, we perform a simple test including the comparison of the ratio of $\frac{M_{New}}{M_{Trained}}$ with the relative magnitudes of the spectrograms, where M_{New} is the spectrogram from data under investigation and $M_{Trained}$ is the expected form of the spectrogram based on past behaviour. First, we present the trained and the anomalous spectrograms for visual inspection.

Figure 6 presents the trained and injected data spectrograms. From a visual inspection, it is easy to identify regions of similarity and difference. According to these results, we were able to detect the DDoS attack, which affected the CPU usage and the number of incoming bytes. As shown in Fig. 6, we can see that the anomalous behaviours in the three metrics occurred at the same time point. To deeply analyse the location of the anomaly, Fig. 7 presents the ratio of the trained and injected data spectrograms. This further confirms the consistency of the anomalous behaviour across the multiple metrics.

The injected anomaly has successfully been detected in each of the presented metrics. However, given the symmetry of the anomaly shape further explanation is required.

Wide side-bands are seen at lower frequencies (frequencies ≤ 0.004); this is because the wavelet transform is based on the FFT and therefore expects periodic data. Cloud-based data can contain periodic behaviour, as the use of cloud-based services is user-need driven. In the data monitoring and collection,

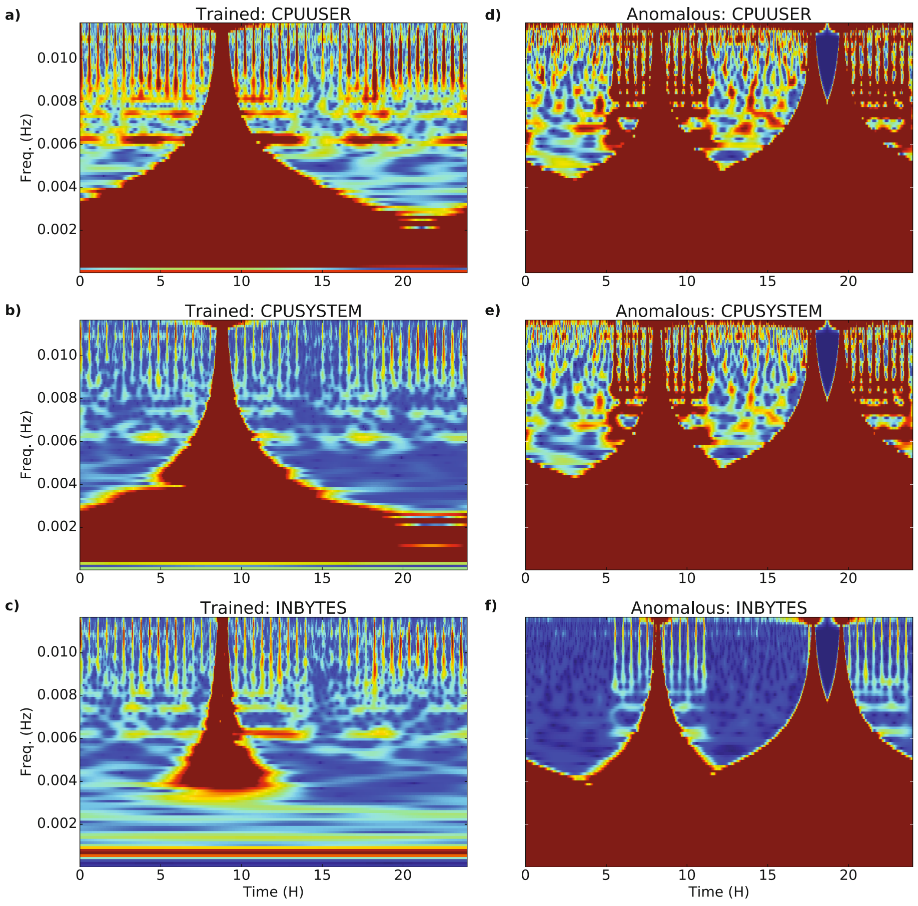


Fig. 6. Visual comparison of trained spectrograms (left) and spectrograms that contain a simulated DDoS attack (right).

similar loads were observed on the monitored VMs from one day to the next. This is appropriate as it reflects a real-world scenario with periodic user behaviour.

Additionally, the wavelet method employed here is multi-scale. This means that we can reduce time-resolution to increase frequency resolution, and vice-versa. One effect of this means that the side bands of the highlighted anomaly will increase due to reduced time resolution.

Finally, the size of the DDoS attack is also a point for discussion. It is possible that the duration of the simulated attack was unnecessarily large. The simulated DDoS lasted for 100 min and occupied 7% of the total modelled day.

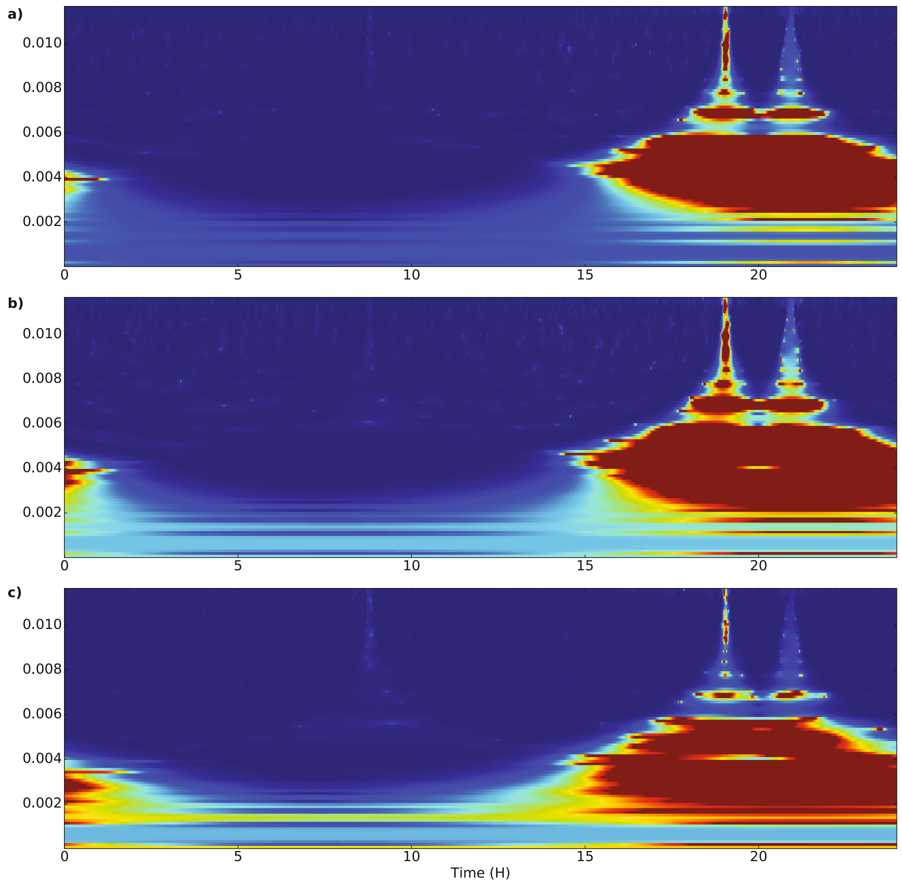
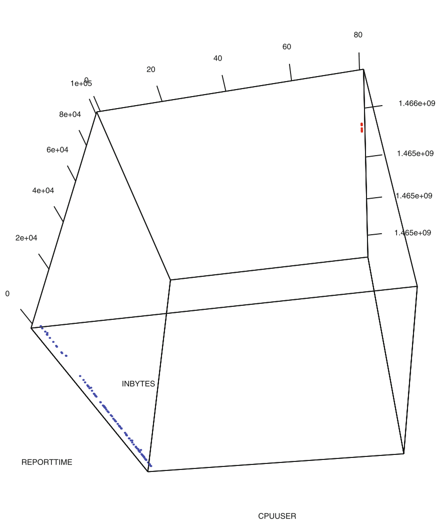


Fig. 7. Plot of the ratio of the anomalous spectrograms **(a)** CPUUSER, **(b)** CPUSYSTEM and **(c)** INBYTES to the trained spectrograms shown in Fig. 6. The high ratios (red) indicate the presence of anomalous behaviour. (Color figure online)

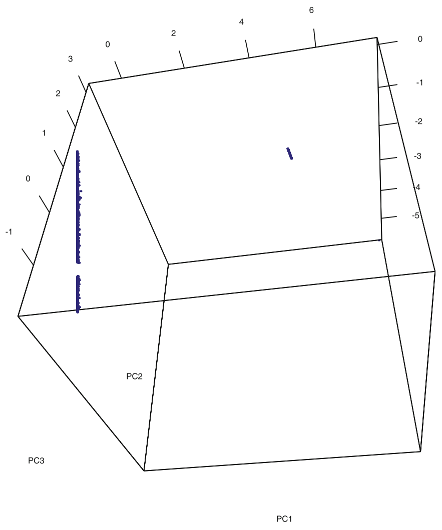
6.4 Principal Component Analysis

In this section, we present comparisons between the results of the wavelet inspired method and a pure statistical approach to show the former's significance. In this case, we calculate the mean of the data column and determine the distance of each element from the centroid. Outliers are determined to be any points that lie outside a confidence interval of 99%.

Figure 8(a) displays two of the injected metrics versus REPORTTIME. The detected anomalies are highlighted in red. This verifies that the injected anomalies satisfy the definition of an anomaly sought by this method (they lie outside the strict confidence interval).



(a) 3D plot of Metrics: REPORTTIME, CPUIDLE and INBYTES.



(b) 3D plot of Metrics: REPORTTIME, CPUIDLE and INBYTES.

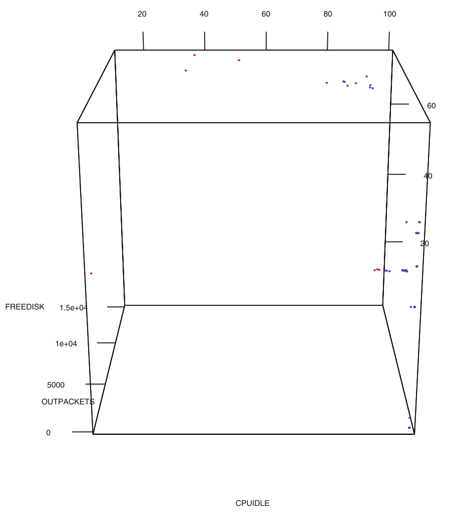
Fig. 8. 3D plots of sample injected metrics and PCA analysis.

Figure 8(b) contains the results of the Principal Component Analysis (PCA). Here it is shown that for the much reduced system, the data is not normally distributed and two of the PCA axes returned would be sufficient to describe the data.

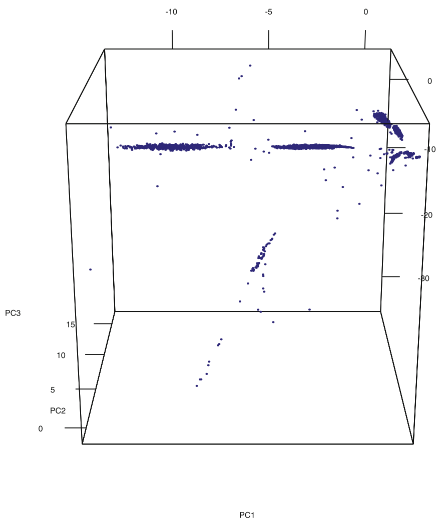
A limitation of this statistical method is that for a given confidence interval, it will always identify points as anomalous (false positives). Another limitation with PCA analysis is that the data supplied are required to be linearly independent (no redundant data) and therefore this requires some knowledge of the data to know which metrics are co-linear. For a low number of metrics, this can addressed quite quickly; however, this can become tedious for higher numbers of metrics recorded.

Figure 9(a) presents a 3D plot of the three analysed example metrics from one of the training data-sets used to train the wavelet-based anomaly detector. The three metrics FREEDISK, OUTPACKETS and CPUIDLE are plotted. From the plot it is clear that anomalous points are identified.

Figure 9(b) shows the PCA reduction performed on 10 metrics. The axes (PC1, PC2 and PC3) are the three most significant vectors demonstrating the extent of the non-normality of the data. Taking a confidence interval of 99% yielded many anomalies. This is expected, as a purely statistical approach will, by construction, always discover anomalous points regardless of whether the points are in fact anomalous or not. Furthermore, the number of outliers will be determined by the confidence interval selected. The interesting thing is that this approach failed to detect some of the injected anomalies in the gathered data. This demonstrates an advantage of the wavelet method over a pure statistical approach.



(a) 3D plot of Metrics: REPORTTIME, CPUIDLE and INBYTES.



(b) 3D plot of Metrics: REPORTTIME, CPUIDLE and INBYTES.

Fig. 9. 3D plots of sample injected metrics and PCA analysis.

7 Conclusion

This paper presented an anomaly detection framework for detecting anomalous behaviour of services hosted on Cloud platforms. It contains a monitoring tool to monitor service executions in Clouds and gather monitoring data for analysis. A wavelet-based detection algorithm was implemented to provide a multi-level analysis of the monitoring data for anomaly detection. It uses frequency domain and time domain information to estimate an anomaly-free spectrogram. The healthy spectrogram is trained (removing seasonality and noise/randomness) by using an extended two-dimensional Welford algorithm to create two-dimensional mean and standard deviations. These quantities are then used to check for the presence of anomalies by comparing the trained mean and standard deviation with those of the new data.

The framework was evaluated based on a Cloud service deployment use-case scenario in an OpenStack evaluation testbed. We used 10 days of gathered monitoring data from the service execution from which a day's data were systematically injected with anomalies for the evaluation. The wavelet-inspired method successfully detected the injected anomalies, and a brief comparison was made with a pure statistical approach, reinforcing the success of our technique.

In the future, we aim to progress this work to a near real-time implementation where the anomaly detection will be carried out on the monitoring data at runtime. The effect of moving to real time will mean the introduction of a time-window, which will be continuously updated as the monitoring platform reports updated metric values. Moving to real time will also distribute the computational workload as each time the metrics are updated, the spectrogram will be appended to rather than being entirely recalculated. Further extensions to this work will permit the comparison of multiple ANNs across multiple (and individual) metrics, which would allow for cross-metric comparison while retaining the ability of identifying the metric(s) containing the anomaly. This will allow for the detection of more complex anomalies in Cloud platforms. Further work is also possible to extend the anomaly injection techniques so we can better determine the limits of this and other anomaly detection frameworks.

Acknowledgements. The research work described in this paper was supported by the Irish Centre for Cloud Computing and Commerce, an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority.

References

1. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: BlinkDB: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European Conference on Computer Systems, pp. 29–42. ACM (2013)
2. Agarwala, S., Alegre, F., Schwan, K., Mehalingham, J.: E2EProf: automated end-to-end performance management for enterprise systems. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, pp. 749–758, June 2007
3. Albanese, D., Visintainer, R., Merler, S., Riccadonna, S., Jurman, G., Furlanello, C.: mlpy: machine learning Python (2012). http://mlpy.sourceforge.net/. Accessed 22 Feb 2016
4. Apache Software Foundation. Apache JMeter (2016). http://jmeter.apache.org/. Accessed 06 Jan 2016
5. Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D., Zhang, M.: Towards highly reliable enterprise network services via inference of multi-level dependencies. In: SIGCOMM. Association for Computing Machinery Inc., August 2007
6. Bakhtazad, A., Palazoglu, A., Romagnoli, J.A.: Detection and classification of abnormal process situations using multidimensional wavelet domain hidden Markov trees. Comput. Chem. Eng. **24**(2), 769–775 (2000)
7. Buzen, J.P., Shum, A.W.: MASF - multivariate adaptive statistical filtering. In: International CMG Conference, pp. 1–10 (1995)
8. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM Comput. Surv. **41**(3), 15:1–15:58 (2009)
9. Doelitzscher, F., Knahl, M., Reich, C., Clarke, N.: Anomaly detection in IaaS clouds. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 387–394, December 2013
10. Emeakaroha, V.C., Brandic, I., Maurer, M., Dustdar, S.: Low level metrics to high level SLAs - LoM2HiS framework: bridging the gap between monitored metrics and SLA parameters in cloud environments. In: 2010 International Conference on High Performance Computing and Simulation (HPCS), pp. 48–54, July 2010
11. Emeakaroha, V.C., Netto, M.A.S., Calheiros, R.N., Brandic, I., Buyya, R., De Rose, C.A.F.: Towards autonomic detection of SLA violations in cloud infrastructures. Future Gener. Comput. Syst. **28**(7), 1017–1029 (2012)

12. Fatema, K., Emeakaroha, V.C., Healy, P.D., Morrison, J.P., Lynn, T.: A survey of cloud monitoring tools: taxanomy, capabilities and objectives. J. Parallel Distrib. Comput. **74**, 2918–2933 (2014)
13. Frigo, M.: A fast Fourier transform compiler. ACM Sigplan Not. **34**, 169–180 (1999). ACM
14. Gander, M., Felderer, M., Katt, B., Tolbaru, A., Breu, R., Moschitti, A.: Anomaly detection in the cloud: detecting security incidents via machine learning. In: Moschitti, A., Plank, B. (eds.) Trustworthy Eternal Systems via Evolving Software, Data and Knowledge, pp. 103–116. Springer, Heidelberg (2013)
15. Guan, Q., Fu, S.: Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In: 2013 IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), pp. 205–214, September 2013
16. Guan, Q., Fu, S.: Wavelet-based multi-scale anomaly identification in cloud computing systems. In: 2013 IEEE Global Communications Conference (GLOBECOM), pp. 1379–1384, December 2013
17. Guan, Q., Fu, S., DeBardeleben, N., Blanchard, S.: Exploring time and frequency domains for accurate and automated anomaly detection in cloud computing systems. In: 2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 196–205. IEEE (2013)
18. Gul, I., Hussain, M.: Distributed cloud intrusion detection model. Int. J. Adv. Sci. Technol. **34**, 71–82 (2011)
19. Hodge, V.J., Austin, J.: A survey of outlier detection methodologies. Artif. Intell. Rev. **22**(2), 85–126 (2004)
20. Ibidunmoye, O., Hernández-Rodriguez, F., Elmroth, E.: Performance anomaly detection and bottleneck identification. ACM Comput. Surv. **48**(1), 1–35 (2015)
21. Lin, M., Yao, Z., Gao, F., Li, Y.: Toward anomaly detection in IaaS cloud computing platforms. Int. J. Secur. Appl. **9**(12), 175–188 (2015)
22. Liu, A., Chen, J.X., Wechsler, H.: Real-time timing channel detection in an software-defined networking virtual environment. Intell. Inf. Manag. **7**(06), 283 (2015)
23. Mi, H., Wang, H., Yin, G., Cai, H., Zhou, Q., Sun, T., Zhou, Y.: Magnifier: online detection of performance problems in large-scale cloud computing systems. In: 2011 IEEE International Conference on Services Computing (SCC), pp. 418–425, July 2011
24. Penn, B.S.: Using self-organizing maps to visualize high-dimensional data. Comput. Geosci. **31**(5), 531–544 (2005)
25. Reynolds, P., Killian, C., Wiener, J.L., Mogul, J.C., Shah, M.A., Vahdat, A.: PIP: detecting the unexpected in distributed systems. In: Proceedings of the 3rd Conference on Networked Systems Design and Implementation, NSDI 2006, Berkeley, CA, USA, vol. 3. USENIX Association (2006)
26. Sha, W., Zhu, Y., Chen, M., Huang, T.: Statistical learning for anomaly detection in cloud server systems: a multi-order Markov chain framework. IEEE Trans. Cloud Comput. (2015). https://doi.org/10.1109/TCC.2015.2415813
27. Song, X., Wu, M., Jermaine, C., Ranka, S.: Conditional anomaly detection. IEEE Trans. Knowl. Data Eng. **19**(5), 631–645 (2007)
28. Videla, A., Williams, J.J.W.: RabbitMQ in Action: Distributed Messaging for Everyone. Manning Publications Company, Grand Forks (2012)
29. Wang, C., Talwar, V., Schwan, K., Ranganathan, P.: Online detection of utility cloud anomalies using metric distributions. In: 2010 IEEE Network Operations and Management Symposium (NOMS), pp. 96–103, April 2010

30. Wang, C., Viswanathan, K., Choudur, L., Talwar, V., Satterfield, W., Schwan, K.: Statistical techniques for online anomaly detection in data centers. In: 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 385–392, May 2011
31. Zhang, Z., Wang, Y., Wang, K.: Fault diagnosis and prognosis using wavelet packet decomposition, Fourier transform and artificial neural network. J. Intell. Manuf. **24**(6), 1213–1227 (2013)