Automated Anomaly Detection and Root Cause Analysis in Virtualized Cloud Infrastructures

Jieyu Lin, Qi Zhang, Hadi Bannazadeh, Alberto Leon-Garcia
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada
Email: {jieyu.lin,ql.zhang,hadi.bannazadeh,alberto.leongarcia}@mail.utoronto.ca

***Abstract*—Cloud data centers today use virtualization technologies to facilitate allocation of physical resources to multiple applications. As cloud data centers continue to grow in scale and complexity, effectively monitoring and identifying system anomalies is becoming a critical problem. Furthermore, due to complex dependencies between system components in a virtualized data center, a single cause of anomaly can typically trigger multiple alarms. Therefore there is also a need to efficiently analyze and identify the causes of the anomalies in a scalable and effective manner, in order to reduce the overhead of diagnosis and troubleshooting performed by the cloud operator.**

Motivated by these observations, we present a mechanism for automatic anomaly detection and root cause analysis in virtualized cloud data centers. We first use unsupervised learning techniques to identify abnormal system behaviors, and then propose a technique for root cause analysis with consideration to anomaly propagation among system components. Using a real virtualized cloud testbed, we show that our mechanism efficiently identifies system anomalies and accurately determines their causes.

I. Introduction

Businesses are increasingly adopting the cloud computing model to manage their service infrastructures. As a result, cloud data centers today host a wide variety of applications ranging from user-interactive services to background analytics jobs. Virtualization technologies are widely used to facilitate the allocation of physical resources to multiple service applications.

As cloud data centers continue to grow in scale and complexity, effectively monitoring and managing cloud data centers becomes a critical challenge. In particular, anomaly detection is a major objective in cloud monitoring that aims to identify abnormal network and system behaviors such as operator errors, hardware and software failures, resource over- and under-provisioning. Given the ever-increasing scale coupled with the high complexity of software, applications and workload patterns, anomaly detection methods must operate automatically at run time without the need for prior knowledge about normal or anomalous behaviors. Furthermore, they should be sufficiently general to support multiple levels of abstraction and subsystems, and handler the different metrics used in large-scale systems.

While network and system anomaly detection has been a subject of extensive study, we found existing solutions do not fully meet the above design requirements. For example, many existing techniques only apply to the single metric case, where only a single metric (e.g. CPU utilization) is considered for anomaly detection (e.g. [14][15]). Furthermore, because cloud applications often show fluctuating resource demand (e.g. operating under the time-of-the-day effect) that typically follow multi-modal distributions, statistical anomaly detection techniques (e.g. [17]) that assume a pre-defined (e.g. Gaussian) distributions do not work well for our case. In addition, as cloud infrastructures typically host tremendous numbers of applications, the monitoring infrastructure must be highly scalable and efficient. Anomaly detection techniques that incur high computational cost or storage requirements are hence not suitable for anomaly detection in cloud infrastructures. Lastly, due to complex dependencies between system components in a virtualized data center, a single cause of anomaly can typically trigger multiple alarms. Therefore designing a scalable system that can efficiently analyze and identify the causes of the anomalies is important in reducing the overhead of system diagnosis and troubleshooting.

Motivated by the above limitations, in this paper we propose a new anomaly detection system for virtualized cloud data centers. We leverage unsupervised learning techniques for identifying anomalies at component[1]-level. Given the anomaly detected, we propose a solution that can quickly identify the source of the anomaly in the system. To achieve high scalability and efficiency, we have implemented our framework using Apache Spark [1]. In experiments using a real cloud testbed running large numbers of applications, we show our mechanism can efficiently identify system anomalies and accurately determine the cause of the anomaly in the system.

The paper is organized as follows. In Section II we provide a motivating example to illustrate the challenge of anomaly detection in virtualized data centers. We present the architecture of our system in Section III, and describe our anomaly detection algorithm in Section IV. Section V presents our proposed algorithm for root cause analysis. We describe the system implementation in Section VI and provide our experiment results in Section VII. Lastly, we survey related work in Section VIII and conclude the paper in Section IX.

[1]We define a component as a entity at different levels of the software stack. For example, A physical machine, a switch and a link are physical layers components, whereas VMs are considered virtual layer components

978-1-5090-0223-8/16/$31.00

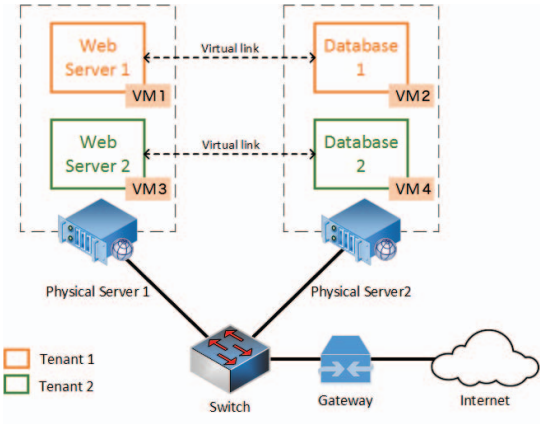


Figure 1: Motivating Example

II. A Motivating Example

First, we provide an example to illustrate how automatic anomaly detection and root cause analysis can reduce management and diagnosis overhead. For the sake of simplicity, we focus on a small part of a large cloud infrastructure (Figure 1) that consists of two physical servers, a network switch, and a gateway within a large data center. Each physical server is connected to the switch through a physical link. The gateway is connected to the switch to provide Internet access.

We assume there are two tenants in this environment, each of which runs a 3-tiered web application on VMs that are hosted by the two physical servers. Each user has two VMs. In both cases, the web server and application server are co-located in one VM and the database is located in another VM. Managed by a VM scheduler, these four VMs are placed so that the two web servers VMs are hosted in physical machine 1 and the two database server VMs are hosted in physical machine 2. Initially, the two tenants are satisfied with the resource allocation as their own service requests are serviced by the shared resources in the virtualized data center.

Subsequently, a web hacker attacks the web application of tenant 1 by launching a DDoS attack. As a result, a large amount of packets are sent to the web server 1, which causes bandwidth contention that decreases the web server processing rate and the number of database lookups. Due to the large volume of traffic in the attack, unexpectedly, the web application of the other tenant is affected. The reason is that the web server 2 and web server 1 share the same physical network link for database access and Internet access. Since the attack almost saturates the physical link, it affects the performance of the tenant 2's web application as well. The output of the monitoring system is shown in Figure 2. In such a shared environment, tenant 2 can see the performance problem of his web application, but he has no clue about the cause. At this point, the cloud provider should be responsible for handling this issue, but the existing monitoring system can not identify the root cause automatically. In practice, due to the scale of the infrastructure, the problem tracing can be even more difficult.

If anomaly detection and root cause analysis can be executed in real time automatically, the system would be able to send alarm to the administrator and suggest the web server 1 and its input traffic as the root cause. Therefore, we believe anomaly detection and root cause analysis should go hand-in-hand in order to simplify the process of diagnosis and troubleshooting.

III. System Description

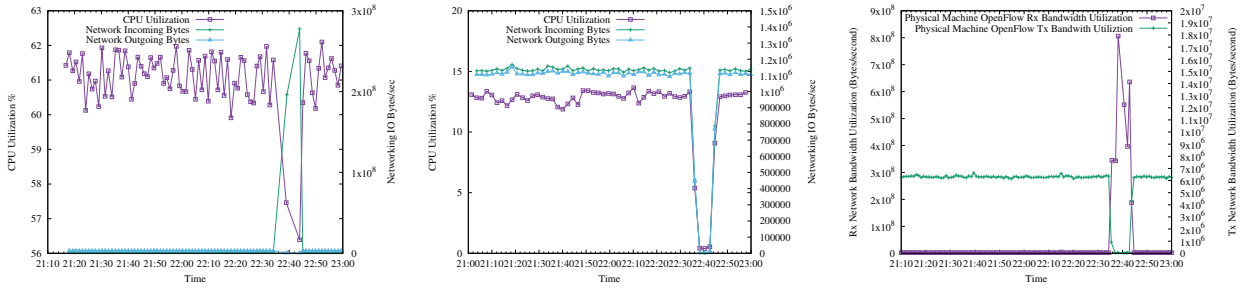
In order to provide integrated anomaly detection and root cause analysis, in this section we present a system that can jointly achieve both objectives. The system contains two main modules: Component Anomaly Detectors module and Anomaly Correlation Engine. Given the set of metrics collected for the components being monitored, the *Component Anomaly Detectors* identify misbehaving components (i.e. components that show abnormal behaviors) and raise alarms once they are detected. In our current implementation, there is one component anomaly detector for each system component. To identify anomalies, each component anomaly detector builds a profile for its corresponding component through historical traces. At run-time, each component anomaly detector checks whether the current measurements of a component deviates from the profile of predicted values. If so, an alarm is raised to indicate the component is experiencing an anomaly.

Once a set of anomalies have been identified at a given time, the information regarding the anomalies are sent to the *Anomaly Correlation Engine*, which tries to identify the source of the anomalies through root cause analysis. To achieve this objective, the Anormaly Correlation Engine relies on domain knowledge captured by the current network and system configurations. This includes data center topology, server and network configurations, as well as virtual machine and application characteristics and settings.

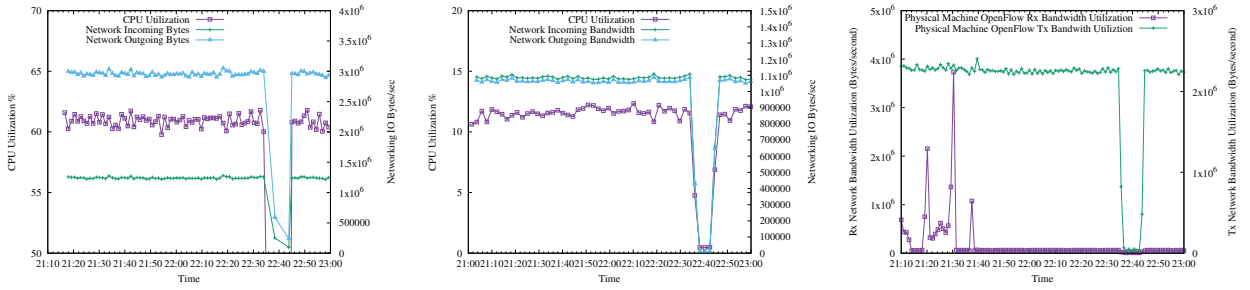
In the next two sections, we will provide the detailed design of a Component Anomaly Detector, followed by describing the implementation of Anomaly Correlation Engine.

IV. Component Anomaly Detector Design

In this section we discuss the detailed design of a component anomaly detector. Specifically, the design of a component anomaly detector must satisfy the following requirements: 1) **Handling unlabeled measurement data:** A measurement is *labeled* if it has been correctly identified as either normal or abnormal. In practice, getting labeled measurement data is extremely difficult due to the large variety of applications and components running in the cloud, as well as the intrinsic dynamism (e.g. demand fluctuation) inherited in these systems. To keep our approach general, we require the anomaly detection mechanism to work with unlabeled data. This implies that we must use unsupervised learning algorithms for anomaly detection. 2) **No assumption on underlying distributions:** Many anomaly detection algorithms assume the measurement data follow specific (e.g. Gaussian) distributions. This assumption generally does not hold for cloud environment where workloads can fluctuate significantly over time. 3) **Scalability and efficiency:** Given the large number of applications and



(a) Web Server 1 VM CPU and Network Utiliza-(b) Database 1 VM CPU and Network Utilization (c) Physical Machine 1 Network Utilization
tion



(d) Web Server 2 VM CPU and Network Utiliza-(e) Database 2 VM CPU and Network Utilization (f) Physical Machine 2 Network Utilization
tion

Figure 2: Resource Utilization of VMs and PMs

components to be monitored, our anomaly detection mechanism must be highly scalable. This detection algorithm must incur both low space and time complexity. Many of the existing anomaly and outlier detection algorithms, such as k-nearest neighbor search (kNN) [8] must keep large volumes of historical data in memory, which make them unsuitable for our purposes.

To satisfy these requirements, we decided to use an unsupervised learning algorithm on historical traces. This problem can be modelled as follows: Let C denote the set of components that comprise the cloud system, and let T denote the duration of the historical traces being considered. For each component $c \in C$, we assume at a given time t there a set of measurements captured by a vector $\mathbf{x}_t^c = \{x_{1t}^c, x_{2t}^c, ...x_{n_c t}^c\}$ that captures the state of component c at time t, where n_c is number of metrics being monitored for c. Assuming that component c operates in normal conditions most of the time during time period T, our goal is to use an unsupervised learning algorithm to create a model that best captures all the observations in $\mathbf{x}_t^c$ for $1 \leq t \leq T$.

While there are many potential learning algorithms (e.g. Statistical Testing [10], Hidden Markov Model [6]) that we can use, we do not restrict the algorithm that is used for individual components. In our implementation we use K-means clustering due to its simplicity and its intuitiveness. Specifically, we are then given a set of historical measurements $\mathbf{X}^c = \{\mathbf{x}_1^c, \mathbf{x}_2^c ... \mathbf{x}_T^c\}$, that can be mapped to a n_c-dimensional space. We first normalize each measurement $\mathbf{x}_t^c = \{x_{1t}^c, x_{2t}^c, ...x_{n_c t}^c\} \in \mathbf{X}^c$ using standard z-score measures. Specifically, we create normalized measurement $\bar{\mathbf{x}}_t^c = \{\bar{x}_{1t}^c, \bar{x}_{2t}^c, ...\bar{x}_{n_c t}^c\}$ by:

$$\bar{x}_{it}^c = \frac{\bar{x}_{it}^c - \mu_i^c}{\sigma_i^c}$$

where μ_i^c and σ_i^c denote the average and standard deviation of dimension i for component c. The K-means clustering algorithm aims at finding K centroids $\{\bar{\xi}_1^c, \bar{\xi}_2^c, ...\bar{\xi}_k^c\}$ in a n_c dimensional feature space and assigns each measurement vector $\mathbf{x}_t^c$ to one of the centroids. Let N_k denote the measurement vectors assigned to centroid $\bar{\xi}_k$, the goal of K-means algorithm is to minimize the following similiarity score:

$$score = \sum_{i=1}^{k} \sum_{j \in N_k} ||\bar{\xi}_i^c - \bar{\mathbf{x}}_j^c||^2$$

where $||a - b||$ denote the Euclidean distance between two points a and b in the feature space. To determine the value of K, we adopt a common approach which is to pick the value of K such that increasing the number of clusters from K to $K+1$ does not achieve significant gain in terms of minimizing $score$. Furthermore, if a centroid contains only a single element, we remove the centroid from our solution.

Once we have obtained the centroids, we specify a threshold θ that is used to separate normal measurements from the abnormal observations. Specifically, for each centroid i, we compute the mean $\bar{\mu}_i^c$ and standard deviation $\bar{\sigma}_i^c$ of the Euclidean distances between the points N_i and centroid i.

During live detection, we use a sliding window (i.e. looking at the latest few minutes of data) to obtain more stable

monitoring input data. For each newly arrived measurement $\mathbf{x}_j^c$ in the sliding window, we compute the anomaly intensity of the measurement as the ratio between the normalized Euclidean distance from $\mathbf{x}_j^c$ to its nearest centroid $k_{nearest}$ and θ:

$$I(\mathbf{x}_j^c) = \frac{||\mathbf{x}_j^c - \bar{\xi}^c_{k_{nearest}}|| - \bar{\mu}^e_{k_{nearest}}}{\theta \cdot \bar{\sigma}^c_{k_{nearest}}}$$

Denoting the max distance between a training data to its centroid in a cluster as d_{max}, we define $I_{max}(\mathbf{x}_j^c)$ as:

$$I_{max}(\mathbf{x}_j^c) = \frac{||\mathbf{x}_j^c - \bar{\mu}^c_{k_{nearest}}||}{d_{max}}$$

For $I(\mathbf{x}_j^c)$, if we select θ to be 2 so we would be able to have an accuracy of around 95% in a Gaussian distribution. $I_{max}(\mathbf{x}_j^c)$ is useful for detecting a serious anomaly or non-Gaussian distribution. If $I_{max}(\mathbf{x}_j^c) > 1$, then we know that this is a serious anomaly since this distance is higher than any training points we have seen. On the other hand, if $I_{max}(\mathbf{x}_j^c) > I(\mathbf{x}_j^c)$, then it provides a hint that the cluster is not Gaussian distributed. These two parameters guide our decision in anomaly detection.

If anomalies are detected, further analysis is conducted to better characterize each anomaly. We would like to understand how the measurement point deviates from the nearest centroid $k_{nearest}$ in each dimension. We do that in the profiling phase.

V. Root Cause Analysis

The technique presented in the previous section allows us to identify misbehaving components, i.e., the components that show abnormal behaviors. However, as mentioned previously, a single anomaly in the system can often cause multiple components to misbehave. To facilitate the diagnosis and troubleshooting process, we introduce an automatic root cause analysis technique to identify the component that is most likely to be the source of the observed anomalies.

The main challenge in root cause analysis is to identify the correlation between abnormal system observations. Due to the complex dependencies between components in virtualized clouds, we need a way to capture the causal relationship between observed anomalies in the system based on our knowledge of the network and system configurations and their resource consumptions.

A. Anomaly Propagation Path

To perform root cause analysis, we need to understand how an anomaly can propagate from component to component. We identify two types of anomaly propagation paths:

Vertical propagation refers to performance interferences due to virtualization and VM collocations. There are three possible scenarios: (1) an anomaly of a physical machine (PM) causes a VM to misbehave, (2) an anomaly of a VM causes a PM to misbehave , and (3) two VMs collocated in the same PM interfere with each other. Notice that even though we may consider the third scenario as a combination of the first two scenarios (i.e. a VM causes a PM to be behave differently, which in turn affects another VM), doing so requires we consider the PM as a misbehaving component, which may not always be the case. For example, consider a scenario where a PM is constantly operating at near maximum outgoing bandwidth utilization, while one VM starts consumption significant bandwidth usage, which in turn cuts down the bandwidth utilization of another VM. In this case, the PM is behaving normally, while the anomaly of one VM still affects the other VM. This example illustrates that we have to consider the third scenario as a separate anomaly propagation scenario.

Horizontal propagation refers to anomaly propagation due to network communications of the same application. For example, when a web application is experiencing a Denial-of-Service attack, the high request rate of the application tier may cause the database tier to behave erroneously. We require both source and destination of the anomaly propagation to show abnormal behavior simultaneously. While there are cases where anomalies may propagate without meeting this requirement, in general these cases are very rare, yet difficult to be analyzed automatically. Therefore, our system would report multiple, separate anomalies instead of reporting anomalies originating from a single source.

We use the motivating example to illustrate the concept of anomaly propagation paths. In the motivating example, anomaly in Web Server 2 VM is caused by performance interference from Web Server 1 VM, and the anomalies in Database 1 VM and Database 2 VM are the results of horizontal propagation caused by communications with Web Server 1 VM and Web Server 2 VM respectively.

We now consider these two types of propagations separately, and identify the conditions under which anomaly propagations may occur. We try to adopt a conservative approach, namely, we say an anomaly propagation has occurred only if it is supported by a strong evidence. Otherwise, we may treat them as separate, uncorrelated anomalies.

B. Vertical Propagation

In vertical propagation, the anomalies of a virtual component may propagate to the physical component and vice versa. We consider 3 sub-cases:

1. A VM propagates anomalies to a PM: A VM may propagate anomalies to a PM through excessive resource usage. This implies that both PM and VM are consuming significant resources of a single type. For example, a VM may consume a large amount of memory, causing the PM to show low memory availability. To detect this condition, we first check whether the total VM resource usage adds up to the total PM resource usage, and then identify a single resource for which both PM and VM show usage deviation from their normal conditions by $x\%$.

2. A PM propagates anomalies to a VM: This refers to the cases where the PM is either experiencing a hardware failure or the Operating system / Virtualization software of the PM is malfunctioning. In these cases, both VM and PM must be identified as misbehaving. There are two possible cases: (1) The PM resource usage is much higher than the sum of VM utilizations by a certain percentage (configured based on the

implementation) for a particular resource (e.g. CPU, memory disk, bandwidth), then we say it is most likely the PM is causing the VMs to misbehave. (2) If the PM resource usage is very low, causing all VMs to have resource usage much lower than their typical values, then we also say that it is likely the PM is causing the VMs to misbehave. An example of this is network adapter failure, where network readings for both PM and VMs are zero. As both VM and PM are misbehaving and VM bandwidth usage is lower than usual, we may deduce that the PM is causing the anomaly in the VM.

3. Performance interference between VMs: This happens when there is a resource for which the PM is at maximum usage, while certain VMs show significant increases in resource utilization and others show decreases in resource utilization. For example, if one VM is consuming high bandwidth usage, the other VMs on the same PM may experience low network performance.

In all other cases, we assume there is no vertical propagation, i.e., the anomalies in the collocating components are independent and do not affect each other.

C. Horizontial Propagation

In horizontal propagation, a misbehaving component can propagate its anomaly to its adjacent components through network communications. However, given two misbehaving components that communicate with each other, it is difficult to determine which component is affecting the other. Therefore, by default, we assume the all horizontal propagation paths are bi-directional. However, there are cases where we may clearly identify the propagation paths. Specifically, when two VMs communicate with each other, the network traffic follows a specific pattern (i.e. one VM sends its request to another, while the other responds to the request). Even though in practice it is hard to guess which VM is the sender, we can monitor the ratio between the bandwidth usage between VMs. If the ratio deviates significantly from the normal behavior, we may infer that the VM whose out going traffic becomes significantly lower is the cause of the anomaly.

D. Analyzing Propagation Graphs

Once we have identified the anomaly propagation paths, we can construct a set of *anomaly propagation graphs*. Formally, an anomaly propagation graph is a direct graph that consists of a source component c, and a graph $G^c = (V^c, E^c)$, where V^c is a set of misbehaving components that can be reached from c through propagation paths, and E^c are the set of links that captures all the propagation paths. A propagation graph may contain cycles. For example, a bi-directional horizontal propagation path forms a loop in the propagation graph.

In our system, for each entity, we construct an anomaly propagation graph that represents the possible propagation of anomalies in the system. Therefore, the root cause detection problem becomes finding a set of misbehaving entities whose propagation graphs best capture all the observed anomalies. We use a ranking method for selecting the best propagation graph.

For each anomaly propagation graphs, we assign a rank based on their scores and display it to the operator. The score function of a propagation graph G^c is the minimum total distance from the source anomaly entity to all other entities:

$$score(G^c) = \sum_{v^i \in V^c} d_{v^c, v^i} \quad (1)$$

where d_{v^c,v^i} denotes the shortest propagation distance from the source entity v^c to entity v^i. When an entity v^i in the propagation graph is not accessible by the another entity v^c, the distance between these two entities is considered as infinite. By using this method, there are two other cases we need to handle: 1) **Score Tie**: multiple best propagation graphs exist and their scores are not infinite. We handle this by choosing the propagation graph that has the highest anomaly seriousness[2]; 2) **Infinite Score**: all the propagation graph have infinite scores. In this case, we rank the propagation path by the coverage of the components.

VI. Implementation

We implemented our anomaly detection and root cause analysis algorithms in MonArch [9] which uses the Apache Spark framework to process multi-layer monitoring data. There are three modules to this implementation: (1) Offline component behaviour profiling, which is responsible for processing historical monitoring data to profile each component for the purpose of capturing the normal behaviour; (2) Online anomaly detection, which performs real-time processing of the new monitoring data and determines if each component is in an anomalous state by comparing to the offline profiling results, and (3) Root cause analysis, which analyzes anomalies and hints from online anomaly detection module, this module tries to locate the components that are causing the anomalies.

VII. Experiments

We have evaluated the performance of our system using the SAVI[7] Testbed. We use the scenario shown in Figure 1 as example to demonstrate its effectiveness.

In the web servers, a simple Tornado[3]-based web application is setup with one API that allows accessing of data stored in the database. PostgreSQL [2] is used as the database and is installed in the database servers. To simulate user activities, we created a user simulator that sends HTTP request to the web server with a variable frequency.

We kept this setup and allows it to run for a few days. We then ran the anomaly detection training algorithm on the historical data to learn the normal behaviour of these two web applications. Real time anomaly detection is then launched to detect any anomalies in these two applications. To simulate the DoS attack, we send large amount of large volume of UDP and TCP packets to the web server 1 for 1000 seconds. Figure 2 shows the monitoring data of the VMs and PMs before, during and after the attack.

[2]Defined by the Euclidean distance from the anomaly data point to the nearest centroid

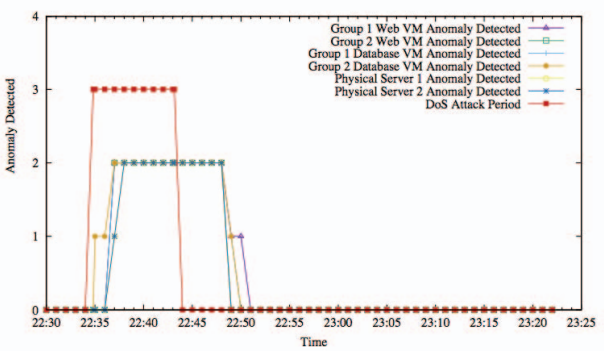


Figure 3: Real Time Anomaly Detection Result - 1 denotes light anomaly, 2 denotes serious anomaly, and 3 denotes when the attack is running

The real-time automatic anomaly detection results from the system are shown in Figure 3. To better evaluate the accuracy of anomaly detection and the detection delay, we overlay the actual attack time in this figure shown by line DoS Attack Period (recorded when we launch and end the attack). It can be seen that the our system can detect all six anomalies with short delay. The start and end of the attack are both captured in the anomaly detection results. In this experiment, the first anomaly is detected for one VM with a 11 second delay, and all 6 anomalies are detected as anomalies with a delay of 2.2 minutes. In this setup, the monitoring data is collected by the MonArch Agents every minute, so detection delay is reasonable with respect to the monitoring data collection rate. We can improve it by increasing monitoring frequency. Moreover, we keep a window of 5 minutes of new monitoring data and use the average value as input for anomaly detection. This explains the relatively higher delay between stop of attack and stop of the anomaly alarms. The window size can be adjusted based on the sensitivity and delay requirements.

To better understand the cause of the anomalies, we run the root cause analysis algorithm discussed in Section V. The propagation graph ranking result from the root cause analysis algorithm is shown in Table I. The highest ranking propagation aligns closely with our expectation and the real cause – the DoS attack that causes the application 1 to behave abnormally and the bandwidth contention causes abnormal behaviour in application 2. By using this root cause analysis results, administrator or management modules can look into the root cause node (i.e. the Web Server 1 VM in this case) and find out that the cause of the problem is due to large amount of traffic generated by DoS attack from a certain source. Then administrator can block that source to prevent attack.

VIII. Related Work

Anomaly detection is a well-studied research topic. There has been extensive study of the problem under many different settings using various statistics tools and machine learning algorithms [5]. However, in the context of virtualized data centers, the problem is still relatively new. For anomaly detection algorithms, the EBAT system uses an entropy-based approach

Ranking	Propagation Path	Score
1	VM 1 –v–> VM 3 –h–> VM 4 VM 1 –h–> VM 2 –v–> PM 2 VM 1 –v–> PM 1	7
2	VM 2 –v–> Physical Machine 2 VM 2 –h–> VM 1–v–> PM 1 VM 2 –h–> VM 1–v–> VM 3 –h–> VM 4	10

Table I: DoS and Resource Contention Use Case Root Cause Analysis Propagation Graph Ranking (v: vertical propagation, h: horizontal propagation)

to detect change in statistical distribution of input metrics. The authors demonstrated that it outperforms threshold-based anomaly detection algorithms [14]. PREPARE uses Tree-Augmented Naive (TAN) Bayesian network to predict online anomalies and proactively take prevention actions [13]. However, these algorithms mainly work at application level, and have not considered the issue of root cause analysis.

For root cause analysis, Sherlock [4] gives an anomaly detection system that uses Bayesian networks to infer the root cause of anomalies. However, it is a supervised learning approach that requires extensive training. FChain [11] use Fast Fourier Transform (FFT) to learn the burstiness of the input signal and use the information to raise alarms, and similar to us, considers anomaly propagation. However, it uses timestamp information to infer the original cause of the anomaly, which may not be accurate in a distributed system where time on each machine may be desynchronized. CloudPD [12] uses k-nearest neighbor for anomaly detection and metric correlation for anomaly classification. However, this approach does not consider anomaly propagation. Finally, the authors of [16] proposed a technique for root cause analysis in component-based systems. However their approach focuses at application-level anomaly correlation, and does not consider the effect of network and system virtualization.

IX. Conclusion

Due to the scale and complexity in a virtualized data center, identifying system anomalies and determining the root cause in a scalable and effective manner is important for system management efficiency. In this paper, we presented a mechanism for automatic anomaly detection and root cause analysis in virtualized cloud data centers. We utilize unsupervised learning techniques to identify abnormal system behaviors, and propose a root cause analysis technique using anomaly propagation among system components. Using a production virtualized cloud testbed, we show that our mechanism can efficiently identify system anomalies and accurately determine their causes in the system.

References

[1] Apache Spark. http://spark.apache.org/.
[2] PostgreSQL. http://www.postgresql.org/.
[3] Tornado web framework. url http://www.tornadoweb.org/en/stable/.
[4] Paramvir Bahl et al. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *ACM SIGCOMM*, 2007.
[5] Varun Chandola and Others. Anomaly detection: A survey. *ACM computing surveys*, 2009.

[6] Zhenhuan Gong et al. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Mgmt. (CNSM) Intl. Conf. on*, 2010.
[7] Joon-Myung Kang, Hadi Bannazadeh, and Alberto Leon-Garcia. Savi testbed: Control and management of converged virtual ict resources. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 664–667. IEEE, 2013.
[8] James M Keller et al. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, 1985.
[9] Jieyu Lin et al. Monitoring and measurement in software-defined infrastructure. In *IEEE IM*. IEEE, 2015.
[10] Constantine Manikopoulos and Symeon Papavassiliou. Network intrusion and fault detection: a statistical anomaly approach. *Communications Magazine, IEEE*, 40(10):76–82, 2002.
[11] Hiep Nguyen et al. Fchain: Toward black-box online fault localization for cloud systems. In *IEEE ICDCS*, 2013.
[12] Bikash Sharma et al. Cloudpd: problem determination and diagnosis in shared dynamic clouds. In *IEEE DSN*, 2013.
[13] Yongmin Tan et al. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *IEEE ICDCS*, 2012.
[14] Chengwei Wang, Vanish Talwar, Karsten Schwan, and Parthasarathy Ranganathan. Online detection of utility cloud anomalies using metric distributions. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 96–103. IEEE, 2010.
[15] Chengwei Wang, Krishnamurthy Viswanathan, Lakshminarayan Choudur, Vanish Talwar, Wade Satterfield, and Karsten Schwan. Statistical techniques for online anomaly detection in data centers. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 385–392. IEEE, 2011.
[16] Kui Wang et al. A methodology for root-cause analysis in component based systems. In *IEEE IWQoS*, 2015.
[17] Nong Ye and Qiang Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(2):105–112, 2001.