Fault Detection for Cloud Computing Systems with Correlation Analysis

Tao Wang, Wenbo Zhang, Jun Wei and Hua Zhong
Institute of Software, Chinese Academy of Sciences
Beijing 100190, P.R. China
{wangtao, zhangwenbo, wj, zhongh}@otcaix.iscas.ac.cn

***Abstract*—The large-scale dynamic cloud computing environment has raised great challenges for fault diagnosis in Web applications. First, fluctuating workloads cause traditional application models to change over time. Moreover, modeling the behaviors of complex applications always requires domain knowledge which is difficult to obtain. Finally, managing large-scale applications manually is impractical for operators. This paper addresses these issues and proposes an automatic fault diagnosis method for Web applications in cloud computing. We propose an online incremental clustering method to recognize access behavior patterns, and uses CCA to model the correlation between workloads and the metrics of application performance/resource utilization in a specific access behavior pattern. Our method detects anomalies by discovering the abrupt change of correlation coefficients with a EWMA control chart, and then locates suspicious metrics using a feature selection method combining ReliefF and SVM-RFE. We validate our method by injecting typical faults in TPC-W an industry-standard benchmark, and the experimental results demonstrate that it can effectively detect typical faults.**

Keywords—Software Monitoring; Performance Anomaly; Fault Detection; Cloud Computing

I. INTRODUCTION

Nowadays, more and more Web applications (e.g., Netflix, Yelp, and Pinterest) are deployed on public cloud computing platforms (e.g., Amazon EC2) to provide Internet-based services. Due to the complexity, dynamism and openness of cloud computing, Web applications are prone to faults which will affect a large population of customers and even lead to severe economic loss. For example, the world's largest e-commerce firm Amazon.com was down for almost 45 minutes in August 2013 due to an unexpected fault, which led to about 5 million dollar loss in transactions. Tellme networks estimates that detecting faults accounts for 75 percent of failure recovery time, and detecting faults in advance prevents 65 percent of failure occurrences [1]. Therefore, fault diagnosis is essential for guaranteeing the reliability and performance of Internet-based services. However, the faults of Web applications (e.g., resource contention, configuration fault, software fault, hardware failure) are always triggered by complex factors in deployment environment at runtime. Since it is difficult to reproduce uncertain faulty contexts (e.g., deadlock caused by concurrency, fault transmission between components), debugging and testing cannot eliminate the contextual faults accounting for about fifteen to eighty percent of detected faults [2]. As the contextual faults is inevitable, fault diagnosis technologies have widely attracted the attention of industrial and academic communities in recent years.

Commercial monitoring tools (e.g., IBM Tivoli, HP OpenView and Amazon CloudWatch) typically require system operators to manually set rules for system metrics. Alerts are raised automatically, when a metric exceeds the pre-defined threshold. However, it is difficult to set suitable thresholds for thousands of metrics in various Web applications. Contemporary fault diagnosis methods for distributed systems model system status during normal operation periods, and detect faults when the monitored status deviates from the built model [3]. However, the large-scale dynamic cloud computing environment has raised great challenges for fault diagnosis. First, since a dynamic deployment context causes application models to change over time, fault diagnosis ought to adapt to the dynamic environment, especially dynamic workloads. Second, since the applications deployed on rented hosts in a cloud computing platform are transparent to system operators, the traditional method of analyzing an application's architecture (e.g., dependencies and interactions between components) is infeasible. Thus, fault diagnosis ought to detect faults and locate root causes without domain knowledge. Third, as various applications are always deployed in a large-scale data center composed of thousands of nodes, system operators cannot afford to manually model the applications with so large amount of metrics collected from many layers (e.g., network, hardware, operating system, virtual machine, middleware, application). Thus, fault diagnosis ought to have the ability of diagnosing faults automatically without manual interventions.

In this paper, we propose an automatic fault diagnosis method for Web applications in cloud computing, to automatically detect faults and locate root causes with statistical monitoring [4]. The contributions of this paper are as follows:

- We propose an online incremental clustering method to differentiate customers' access behaviors. Thus, fault diagnosis in a specific access behavior pattern can effectively improve the diagnosis.
- We use CCA to model the correlation between workloads and metrics automatically without domain knowledge. Our method analyzes system status in multiple layers

(e.g., performance metrics in the application, resource utilization in the system) rather than a single physical layer.

- We detect faults by discovering the abrupt changes of correlation coefficients with a EWMA control chart, and locate root causes with a feature selection method combining ReliefF and SVM-RFE automatically. Our method can be used in various applications without domain knowledge.

II. Workload Characterization and Recognition

A. Workload Characterization

The most typical representative of Web applications is a component-based JEE application, which is usually deployed on a typical multi-tier infrastructure to process transactional requests for online users with Web interfaces. Customers interact with Web applications through a series of HTTP requests; the object of a request is commonly a Web page composed of HTML files generated from Web components (e.g., Java Servlet, Java Server Page and EJB); the application server invokes Web components to process these requests. Customers always access websites to do different operations during different periods in different access sequences as access behavior patterns.

Thus, we first introduce an access matrix to represent customers' access behaviors. Given (*k-1*) components (e.g., servlet) in a Web application, we get a *k*-order state transition matrix in which (*k-1*) states represents (*k-1*) request types and a state represents the state of "Exit", and the probability of state transition during a period describes customers' access behaviors. Let's denote:

$$am = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k1} & p_{k2} & \cdots & p_{kk} \end{bmatrix},$$

where p_{ij} represents the probability of transition from state *i* to state *j*, and $\sum_{j=1}^{j=k} p_{ij} = 1$.

For convenience, we introduce an access behavior vector (*av*) transformed from *am* to characterize customers' access behaviors. Let's denote:

$$av = \{e_1, e_2, ..., e_m, ..., e_{k\times k}\},$$

where $e_m = p_{ij}$, $m = (i-1)\times k+j$, and p_{ij} is an element in *am*.

Furthermore, we introduce a volume vector (*vv*) to characterize customers' access volume. Let's denote:

$$vv = \{c_1, c_2, \ldots, c_m, \ldots, c_{k-1}\},$$

where c_i is the invocation frequency of the i^{th} component, and *k-1* is the number of components.

Thus, we characterize a workload (*wl*) as:

$$wl = <av, vv>.$$

B. Access Behavior Pattern Recognition

After characterizing access behaviors, we ought to recognize customers' access behavior patterns according to the collected sequential vectors *av*s. Our target is to group similar access behaviors into an access behavior pattern, so that the workloads in the same access behavior pattern have similar request sequence and resource demand. Thus, we can use a clustering method to online recognize access behavior patterns by grouping similar *av*s into a cluster, in which an access vector and a cluster present a customer's access sequence and an access behavior pattern (*bp*) respectively.

However, as workloads fluctuate over time, current clustering methods (e.g., *k-means*) suffer limitations in online recognizing workloads. First, the number of clusters is unknown in advance. Second, the method should be robust to outliers by regarding them as anomalies instead of a new pattern. Finally, these access behavior patterns should evolve with time. To address these issues, we propose an incremental clustering method by improving a contemporary clustering method [6], which detects non-spherical clusters and automatically find the number of clusters. It assumes that cluster centers are surrounded by neighbors with lower local density at a relatively large distance from any points with a higher local density.

Algorithm 1 describes the method of updating clusters and recognize access behavior patterns, when a new data point arrives. We first initialize clusters using a clustering method [6] with initial data points. This algorithm finds neighbors from which to the arrival point the distance is less than d_c, and updates the local density of its neighbors (1); increases the local density of every point in the point set by one, and inserts the point whose rank in the local density changes in the update data set (2); updates the minimum distance and significance, and then updates the cluster centers (3-4); adds the arrival point in the corresponding cluster, or takes it as an outlier, if the centers of clusters are not modified (5); reassigns all points, if a point is changed to be a cluster center (6); reassigns the points belonging to the cluster, if a center is changed to be a normal point (7).

Algorithm 1：Incremental Clustering with density peak

```
Input: Data Instance Set S; Cluster Label Set L; Arrival Data Point p
Output: Cluster Label Set L'
1   Find points from which to p the distance is less than $d_c$ to construct
    update data set UDS.
2   For each point k in UDS
      2.1 The local density of point k is updated as: $ld_k = ld_k+1$;
      2.2 For each point j:
          2.2.1  If( $0 \le ld_k - ld_j < 1$ )
                 2.2.1.1  Insert point j in the modified data set UDS;
3   For each point i in UDS
      3.1 Update $md_i$ and $sig_i$;
4   Update cluster centers;
5   If the centers of clusters are not modified
      5.1 Add p in the corresponding cluster, or take p as an outlier;
6   Else if a point i is changed to a center
      6.1 Reassign all of points;
7   Else if a center c is changed to a normal point i
      7.1 Reassign the points belonging to the cluster.
```

We online recognizes access behavior patterns, and characterize system status in a specific pattern. Thus, it can effectively improve the fault diagnosis accuracy by introducing access behavior patterns.

III. Fault Diagnosis

A. System Status Characterization

In our previous work [7], we have used a series of experiments to analyze the correlation between workloads and the metrics of application performance/resource utilization with the TPC-W industry-standard benchmark. The experimental results demonstrate that the access behavior and volume (the number of concurrent users) cause application performance/resource utilization to change.

Thus, we model the correlation between workloads and the metrics of application performance/resource utilization to characterize system status in this subsection. For access behaviors, we use the method proposed in subsection II.B to recognize access behavior patterns. Then, we correlate the access volume (*vv*) and application performance/resource utilization to characterize system status in the specific access behavior pattern (*bp*). There are several challenges in choosing an effective characterization method. First, although some well-known correlation analysis methods (e.g., Pearson) are widely used to model correlations, they only deal with two single variables. However, workloads, resource utilization and application performance are all high-dimension variables. Second, as system runtime status is characterized by many system metrics which correlate with each other, we ought to consider their correlations.

From the experimental results in [7], we observe that it is reasonable for us to assume that:

(1)There are linear relationships between component invocations in a specific access behavior pattern [8]. For example, in the browsing access pattern of a typical e-commerce application complying to the TPC-W specification [8], there are correlations among $r_{select,add}$, $r_{bowse\ select}$ and $r_{home,browse}$ as is denoted:

$$r_{select,add} = 0.3\times r_{browse\ select} = 0.3\times 0.2\times r_{home,browse},$$

where $r_{a,b}$ represents the request probability that customers access component *b* after accessing component *a*. Thus, we use *vv* (volume vector in subsection II.A) to describe workloads.

(2)There are linear relationships between resource utilization metrics in a specific access behavior pattern [3]. For example, a surge of HTTP requests to the front Web server usually leads to a similar surge of SQL requests to the back database server. Thus, the CPU utilization of the Web server and that of the database are also in linearity. Thus, we introduce a resource vector to describe the multiple resource utilization metrics. Let's denote:

$$rv = \{r_1,r_2,\dots,r_i,\dots,r_n\},$$

where r_i is the i^{th} metric, and *n* is the number of resource metrics.

(3)The relationship between response time and throughput^{-1} are also linear. We introduce a performance vector including response time and throughput to characterize application performance. Let's denote:

$$pv = \{rt, \frac{1}{tp}\},$$

where *rt* is the response time and *tp* is the throughput.

Obviously, the metrics of application performance/resource utilization vary with workload fluctuation, so anomalies are detected when the observed metrics cannot be explained by the observed workloads. Therefore, we first model the correlation between workloads and application performance/resource utilization in the normal situation as a baseline. The correlation should keep stable in the normal situation, and thus we detect anomalous status when the correlation breaks down.

Canonical correlation analysis (CCA) is a way of searching linear combinations of the original variables to maximize the correlation between them. Given a workload vector X = (x_1, ..., x_p) and a performance vector Y = (y_1, ..., y_q), CCA find the linear combinations of x_i and y_j, which have maximum correlation with each other [9]. Therefore, we utilize it to discover the correlation between workloads and application performance/resource utilization. Let's denote:

$$S = \{A, B\},$$

where $A=\{a_1,a_2,...,a_m\}^T \in R^{m\times p}$ is the set of workload vectors, $B=\{b_1,b_2,...,b_m\}^T \in R^{m\times q}$ is the set of performance/resource vectors.

Then, we attempt to find the linear combination of the variables in the workload vector *a*, and that in the performance vector *b*. Let's denote:

$$f = u^{T}a\ ,$$

$$g = v^{T}b\ ,$$

We want to find those vectors of $u\in R^p$ and $v\in R^q$, which give us the maximum correlation between the combinations. We get $r_i = corr(f_i, g_i)$, and descend them $r_i \le r_j \le ... \le r_k$. The highly correlated pair (f_i, g_i), which has the highest correlation coefficient, reflects the correlation between workloads and application performance/resource utilization.

As the highest correlation coefficient reflects the correlation between workloads and application performance/resource utilization, we choose it to detect anomalies. With the monitored workload vectors and application performance vectors/resource utilization vectors during operations, we model and update the correlations automatically. We regard the Web application is in an anomaly status, when the correlation coefficient drops significantly.

Note that, since it is computationally expensive to construct system models with thousands of correlated metrics in system resource utilization, and many of them are irrelevant or redundant, we use principal components analysis (PCA) to select representative metrics. PCA uses linear metric combinations called components to demonstrate the correlations among these metrics to reduce the number of variables for analysis. There is almost as much information in the *k* components as in the original *m* metrics, and thus we can use these *k* (*k*<*m*) components to represent the original *m* metrics.

B. Fault Detection

The correlation between workloads and application performance/resource utilization keeps stable in a normal situation, while it fluctuates significantly when a fault is triggered. Since the correlation frequently fluctuates in dynamic networks, the addressed issue is how to decide whether the fluctuation of the correlation coefficient is stable or not. The stability is defined as a state in which the correlation has displayed a certain degree of consistency in the past, and is expected to do so in the future. We monitor the correlation coefficient, and raise an alarm if it is not in stable. For example, the correlation coefficient should keep within a reasonable range in an application server. Resource contention aggregates when a problem (e.g., memory leak) happens, so the throughput cannot probably increase as the number of concurrent user increases. As for the contextual faults, we regard the system status in most cases as normal, and characterize the system healthy situation with the correlation analysis automatically.

Exponentially weighted moving average (EWMA) is a sequential analysis technique for change detection in statistical quality control (SPC). Compared to other abrupt change detection technologies (e.g., wavelet analysis), the EWMA control chart with low computation is suitable for online analysis, and adapts to changing context. The EWMA statistic is calculated as follows:

$$Z_i = \lambda x_i + (1-\lambda) Z_{i-1},$$

where $0 < \lambda < 1$ is the smoothing constant, and Z_i is the EWMA statistic for the i^{th} observation of the correlation coefficient, x_i. The initial value of EWMA statistic z_0 is the average value of the initial correlation coefficients.

The upper control limit UCL_x and lower control limit LCL_x are calculated as follows:

$$UCL_x(i) = \mu_z + L_z \sigma_z$$

$$LCL_x(i) = \mu_z - L_z \sigma_z$$

where L_z is the upper $\alpha/2$ percentage point of the unit normal distribution.

We use the initial data instances to estimate the mean (μ_x) and standard deviation (σ_x) of x_i as follows:

$$\mu_z = \mu_x$$

$$\sigma_z^2 = \sigma_x^2 \left(\frac{\lambda}{2-\lambda}\right)$$

We set $\lambda \in [0.15, 1.0]$ with L_z in range [1.96, 3] corresponding to a level of significance of α=[0.002, 0.05].

If $z_i < \mathrm{LCL}_x(i)$ or $z_i > \mathrm{UCL}_x(i)$, an alarm signal is triggered.

C. Problem location

Faults always cause metrics to fluctuate significantly, so locating anomalous metrics is effective for narrowing down the root cause. After detecting faults with the above method, we label the monitored data instances as positive and negative before and after detecting faults, respectively. As every monitored data instance includes multiple system metrics, our object is to locate the metrics which most probably cause the system status to change from normal to faulty. The more significantly the metric fluctuates before and after detecting the fault, the more probably the metric causes the system status to change from normal to faulty. Thus, the problem of locating suspicious metrics related to the root cause of the detected fault can be abstracted as feature selection.

We label the data instances before detecting the fault as negative, and the data instances after detecting the fault as positive. Let's denote:

$$S=\mathrm{F}(P,N)$$

where S is the suspicious metric set, P is the positive dataset, N is the negative dataset, and F is a feature selection method.

A good feature should differentiate instances in different classes, and have similar instances in the same class. For choosing a feature selection method in locating the suspicious metrics, we should consider the correlations between multiple metrics, which always interfere with each other (e.g., CPU and memory utilization) [3]. We propose a two-phase feature selection method which combines a filter method, ReliefF, to eliminate redundant features, and a wrapper method, SVM-RFE, to rank the left suspicious metrics.

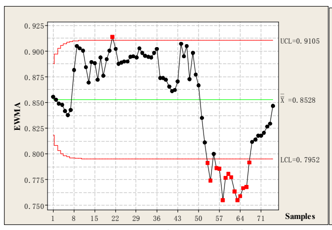
In the first phase, ReliefF first selects a small number of metrics to reduce the number of raw system metrics. ReliefF, which is an efficient method, estimates features according to how well they distinguish near instances [10]. For a given instance, ReliefF searches for two nearest neighbors: one from the same class (called nearest hit) and the other from a different class (called nearest miss). The estimated weight of feature A, W[A], is an approximation of the following difference of probabilities:

W[A]=P(different value of A|nearest instance in a different class)-P(different value of A|nearest instance in the same class).

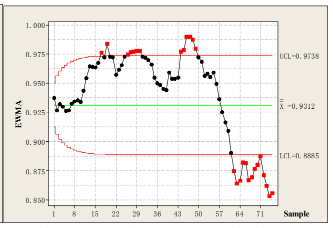
The time complexity of ReliefF is in a linear relationship with the number of features and the number of data instances, so this method introduces limited overhead. Furthermore, this method is able to deal with data instances with dependent and independent features. Therefore, we choose it to eliminate redundant features with low relevance.

Although ReliefF is computationally light and avoids overfitting, the nearest neighbors are defined in the original feature space, which are highly unlikely to be the ones in the weighted space. Furthermore, in the presence of a large number of irrelevant features and mislabeled data instances, the selected subset might not be optimal, in which a redundant subset might be obtained.

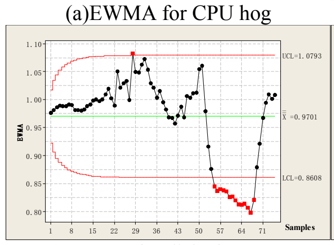
In the second phase, we use SVM-RFE to rank the selected metrics in the first phase. Support vector machine (SVM) has a series of vectors to replace the original data instances for describing the whole dataset. SVM searches an optimal hyper-plane to separate two classes and maximize the distance between them. Kernel functions project the original data into a higher dimensional feature space. Then, we can separate hyper-plane in this feature space with linear methods. SVM-RFE is the generalization of REF in SVM, which eliminates features one by one to obtain an optimal subset of features. SVM-RFE use SVM to find a subset of variables which maximizes the SVM performance. It can overcome the drawback of ReliefF with kernel functions, but creating a new model in each subset evaluation introduces significantly computation overhead. So, we use ReliefF as a preprocessing



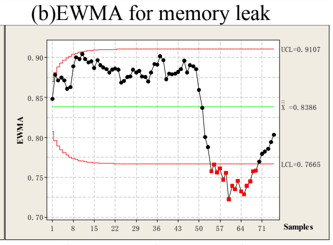
(a)EWMA for CPU hog



(b)EWMA for memory leak



(c)EWMA for disk I/O error



(d)EWMA for network error

Figure 2. Fault Detection with EWMA

step to eliminate a large number of irrelevant features, and then use the SVM-RFE method to rank the left features.

IV. Evaluation

We have applied the above method in our PaaS cloud computing platform which is similar with Google App Engine. We deployed a multi-tier e-commerce benchmark on the platform to validate our method in recognizing access behavior patterns, detecting the injected faults and locating the anomalous metrics related to the faults. We focus on two kinds of faults including system faults (e.g., CPU hog, memory leak, and Disk I/O fault), and performance anomaly because of workload surge.

A. Environment

As is shown in Figure 1, the experiment environment is consisted of three servers, each of which has two Xeon E5 - 2620 2.00GHz CPU with two kernel and 32GB RAM, and runs CentOS 6.4 64-bit with Xen 4.2.0 hypervisors. A server hosts four guest VMs, each of which is assigned with one VCPU and 8GB RAM, and also runs CentOS 6.4 64-bit with Xen 4.2.0. We deploy an open source online e-commerce benchmark Bench4Q [11] complying with TPC-W on our cloud computing platform. The testbed includes a data tier based on a database server MySql, a logic tier based on a application server Tomcat, a load balancer Ngnix and a workload generator. The workload generator simulates dynamic workloads; the application server provides the deployment environment for the application; the deployed application accesses the data source from the database. We collects workload and performance vectors with AspectJ, and monitor system-level metrics from the interface of CentOS.

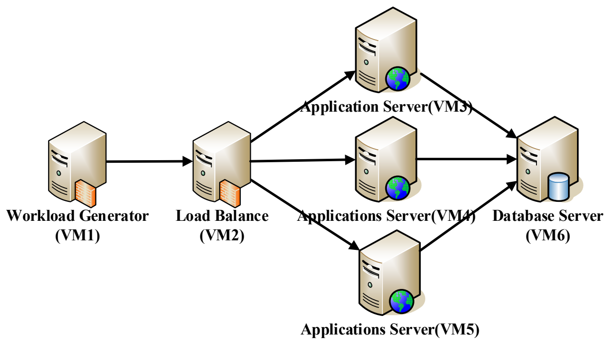


Figure 1 Experiment Architecture

B. Fault Detection

We use a cross-platform monitoring tool Hyperic Sigar to collect monitoring data in Dom0 and guest VMs. Meanwhile, we collect the workload vectors with AspectJ through weaving in the session manager, and calculate the response time and throughput by weaving in the request and response module in Tomcat.

We increase the concurrent users from 4 to 300 with a step number of four in the browsing access behavior pattern, and each experiment lasts about five minutes. We collect the 75 data points, and use CCA to calculate the correlation coefficient between volume vector *vv* and resource vector *rv* for every sample in a sliding window. Then, we use the EWMA control chart to monitor the correlation coefficient in a specific access behavior pattern.

We select some servlets of the application to inject faults at random. The faults are triggered in the 50th sample and eliminated in the 65th sample automatically. During the period with faults, some additional operations are triggered in each interaction with these injected servlets. The CMU report investigates the causes of failures in Web applications, and looks into the prevalence of these failures [12]. Thus, we choose four typical faults analyzed in the report, and employ the method used in existing works [3, 13, 14] to inject them. The faults and injection method are popularly used to validate anomaly detection methods.

The four typical faults to inject are listed as follows:

CPU hog: results from circular wait or endless loops in a program such as the spin lock. We inject this fault in the application by inserting additional computation operations, which are triggered to occupy additional CPU resources.

Memory leak: is due to locating heap memory to objects without releasing them. This fault exhausts the system memory resource gradually, and leads to the system crash eventually. We create an object with the size of *1k* bytes, and referred it to a static variable lest it should be garbage collected by the JVM, when a fault is triggered.

Disk I/O fault: the disk I/O access follows a certain pattern in a specific workload pattern, but the application layer or hard disk failures could also affect the disk access. We simulate such faults by injecting extra operations of reading and writing disk in the application.

Network fault: causes server saturation leading to deny of services, when malicious codes are injected in applications to send diagrams, or the server accepts a large volume of requests from many unknown clients in a network attack. We simulate such faults by injecting malicious codes in the Web application. The application sends UDP packets to any host on the local network, when the injected servlet components are invoked.

As is shown in Figure 3, the experimental results demonstrate that the correlation coefficients are almost in the normal area (between the LCL and UCL) of the EWMA control charts when the faults are not triggered. However, the correlation coefficients are almost in the anomalous area (below the LCL) of the EWMA control charts, when faults are triggered. Note that, alters are not immediately raised, when the faults are triggered; alters are not immediately terminated, when the faults are eliminated. For example, as is shown in Figure 3(c), the disk I/O fault is triggered in the 50th sample, but the first alter is raised in the 53th sample; the fault is eliminated in the 65th sample, but the last alter is raised in the 68th sample. EWMA smooths the sequential correlation coefficients, so a subtle change cannot manifest immediately. Thus, EWMA, which raises alters when continuous multiple anomalies appear, can eliminate occasional coefficient fluctuation.

V. Related work

Many studies model the system behavior including execution paths and component interactions. Chen et al. used a probabilistic context-free grammar to represent the execution paths, in which grammar symbols were components used for servicing requests and grammar rules corresponded to transitions assigned probabilities between components. The paths which failed to be parsed by grammar were regarded as anomalies [15]. Barham et al. used clustering to group paths, and the ones which did not fit the built clusters were anomalous [16]. Chen et al. employed statistics to periodically analyze interactions between one component and the others using χ2-test [17]. These methods are capable of detecting application-level faults. However, they cannot detect the faults caused by resource contention. In this paper, we analyze the system-level metrics which reflect the resource utilization, so our method is able to locate the suspicious resources which probably cause the faults.

Metric correlation based methods characterize the hidden invariant relationships among system metrics, and the anomalies are detected when the relationships are broken. Jiang et al. used autoregressive linear regression with exogenous input (ARX) models to capture the metric correlations [3]. Munawar et al. discussed many linear regression methods to discover metric correlations [18]. Guo et al. investigated Gaussian Mixture Models (GMM) to model the nonlinear correlations between metrics [19]. While the methods are easy to be extended to many applications without domain specific knowledge, it is difficult to model various correlations between so many metrics in complex systems, and the metric correlation changes as the workload pattern evolves [20]. Our method considers the influence of workloads on fault detection by modeling and tracking the correlation in a recognized access behavior pattern.

Some studies pay attention to system performance, which establish a model to predict the expected performance. The predicted performance is compared with the online monitored performance. Such methods require the domain knowledge (e.g., the system internal structure) and accurate parameters (e.g., component service time), which are difficult to obtain in practice. Cohen et al. used TANs to identify which system-level metrics were correlated with the high level performance SLO (Service Level Object) violations [21]. The work aims at finding critical metrics which have an important impact on SLO instead of tracking the system status to detect anomalies. Tan et al. predicted recurrent performance anomalies for virtualized cloud systems by combing attribute value prediction with supervised anomaly classification methods [14]. It employs supervised methods to analyze system-level metrics, so it cannot deal with unseen anomalies. Our method uses a statistical learning method to model the correlation and detects unseen anomalies by analyzing historical data without domain knowledge and fine-grained parameter estimation.

VI. Conclusion

In this paper, we propose an automatic fault diagnosis method for Web applications in cloud computing. We proposes an online incremental clustering method to recognize access behavior patterns, and then uses CCA to model the correlations between workloads and metrics related to the application performance/resource utilization in a specific access behavior pattern. We detects anomalies by discovering the abrupt change of correlation coefficients with a EWMA control chart, and then locates suspicious metrics using a feature selection method combining ReliefF and SVM-RFE. We validate our method by injecting typical faults in TPC-W an industry-standard benchmark, and the experimental results demonstrate that it can effectively detect typical faults. Compared with existing works, we considers the deployment context in fault diagnosis, which improves the accuracy in the dynamic environment of cloud computing. Moreover, we models the behaviors of complex applications automatically without domain knowledge, which is suitable for managing large-scale cloud computing systems. Finally, we can detect faults in multiple layers (e.g., performance metrics in applications, resource utilization in systems) rather than a single physical layer. Although operators can follow these anomalous metrics to narrow down the root causes with our method, they cannot locate faults in a line of source code with the coarse system-level monitoring data. In the future work, we will integrate some fine-granularity methods, such as tracing request and analyzing logs, in our method for locating sections of source code.

Acknowledge·

This work is supported by the National Natural Science Foundation of China under Grant No. 61402450, the Beijing Natural Science Foundation of China under Grant No. 4154088 and the National Key Technology R&D Program under Grant No. 2015BAZ05822.

References

[1] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?," in *the 4th Symposium on Internet Technologies and Systems*, Seattle, WA, 2003, pp. 1-16.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys,* vol. 41, pp. 1-58, 2009.

[3] G. Jiang, H. Chen, and K. Yoshihira, "Modeling and tracking of transaction flow dynamics for fault detection in complex systems," *IEEE Transactions on Dependable and Secure Computing,* vol. 3, pp. 312-326, 2006 2006.

[4] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani, *The elements of statistical learning* vol. 2: Springer, 2009.

[5] A. Williams, M. Arlitt, C. Williamson, and K. Barker, "Web workload characterization: Ten years later," *Web content delivery,* vol. 2, pp. 3-21, 2005.

[6] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science,* vol. 344, pp. 1492-1496, Jun 2014.

[7] T. Wang, J. Wei, F. Qin, W. Zhang, H. Zhong, and T. Huang, "Detecting performance anomaly with correlation analysis for Internetware," *Science China Information Sciences,* vol. 56, pp. 1-15, 2013/08/01 2013.

[8] D. A. Menasc, "TPC-W: A benchmark for e-commerce," *IEEE Internet Computing,* vol. 6, pp. 83-87, 2002.

[9] D. R. Hardoon, S. R. Szedmak, and J. R. Shawe-taylor, "Canonical correlation analysis: An overview with application to learning methods," *Neural Computing,* vol. 16, pp. 2639-2664, 2004.

[10] I. Kononenko, "Estimating attributes: analysis and extensions of RELIEF," in *European Conference on Machine Learning*, Catania, Italy, 1994, pp. 171-182.

[11] W. Zhang, S. Wang, W. Wang, and H. Zhong, "Bench4Q: A QoS-oriented e-commerce benchmark," in *the 35th Annual Computer Software and Applications Conference*, Germany, 2011, pp. 38-47.

[12] S. Pertet and P. Narasimhan, "Causes of failure in web applications," *Parallel Data Laboratory, Carnegie Mellon University, CMU-PDL-05-109,* 2005.

[13] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward, "Efficient fault detection and diagnosis in complex software systems with information-theoretic monitoring," *IEEE Transactions on Dependable and Secure Computing,* vol. 8, pp. 510-522, 2011-01-01 2011.

[14] Y. Tan, H. Nguyen, X. Gu, C. Venkatramani, and D. Rajan, "PREPARE: Predictive performance anomaly prevention for virtualized cloud systems," in *the 32nd International Conference on Distributed Computing Systems*, Macau, China, 2012, pp. 285-294.

[15] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox*, et al.*, "Path-based faliure and evolution management," in *the 1st Symposium on Networked Systems Design and Implementation*, Berkeley, CA, 2004, pp. 23-36.

[16] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *the 6th International Symposium on Opearting Systems Design and Implementation*, California, USA, 2004, pp. 18-31.

[17] H. Chen, G. Jiang, C. Ungureanu, and K. Yoshihira, "Failure detection and localization in component based systems by online tracking," in *the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, Chicago, Illinois, USA, 2005, pp. 750-755.

[18] M. A. Munawar and P. A. S. Ward, "A comparative study of pairwise regression techniques for problem determination," in *Conference of the Center for Advanced Studies on Collaborative Research*, Toronto, Canada, 2007, pp. 152-166.

[19] G. Zhen, G. Jiang, H. Chen, and K. Yoshihira, "Tracking probabilistic correlation of monitoring data for fault detection in complex systems," in *International Conference on Dependable Systems and Networks*, PA, USA, 2006, pp. 259-268.

[20] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward, "System monitoring with metric-correlation models: problems and solutions," in *the 6th International Conference on Autonomic Computing*, Barcelona, Spain, 2009, pp. 13-22.

[21] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in *the 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, 2004, pp. 16-29.