

Combining Supervised and Unsupervised Monitoring for Fault Detection in Distributed Computing Systems

Haifeng Chen Guofei Jiang Cristian Ungureanu Kenji Yoshihira
NEC Laboratories America, Inc.
4 Independence Way
Princeton, NJ 08540, USA
{haifeng, gfj, cristian, kenji}@nec-labs.com

ABSTRACT

Fast and accurate fault detection is becoming an essential component of management software for mission critical systems. A good fault detector makes possible to initiate repair actions quickly, increasing the availability of the system. The contribution of this paper is twofold. First a new concept of supervised and unsupervised monitoring is proposed for system fault detection. We use a statistical method, canonical correlation analysis (CCA), to model the contextual dependencies between system inputs \mathbf{u} and internal behavior \mathbf{x} . By means of CCA, the space \mathbf{x} is transformed into two subsets of variables, which are monitored in a supervised and unsupervised manner respectively. By doing so, our approach can reduce the false alarms resulting from unusual workload changes, and hence achieve high fault detection rate. Second, in order to test the performance of our approach, we simulate a variety of system faults in a real e-commerce application based on the multi-tiered J2EE architecture. Experimental results demonstrate that the CCA based approach can detect injected failures at their early stages when unusual phenomenon is very weak, and hence contribute to enormous time and cost savings in managing large scale distributed systems.

Categories and Subject Descriptors

K.6.4 [Management of Computing and Information Systems]: System Management; I.2.6 [Artificial Intelligence]: Learning

General Terms

Algorithms, Management

Keywords

fault detection, distributed computing, supervised monitoring, canonical correlation analysis, internet services

1. INTRODUCTION

Distributed computing systems are becoming increasingly complex and hard to manage due to the interactions between workload,

software structure, hardware, traffic conditions and so on. Such complexities increase potential for those online services to suffer from various user visible failures. For example, a bug in certain software component of an e-commerce application will cause items not being added to a shopping cart or an error message being displayed. Other types of failures also exist resulting from a wide variety of human's operational errors, hardware and software faults.

There have been many research activities of detecting faults in electrical and mechanical systems and a number of methods have been proposed in those domains. However, some specific features of distributed systems introduce new challenges for the fault detection task. For instance, a large percentage of actual failures in computing systems are partial failures, which only break down part of service functions and do not affect the operational statistics such as response time. Such partial failures cannot be easily detected by traditional tools, such as pings and heartbeats [1]. It is imperative to have more advanced techniques to cope with those failures.

Meanwhile the statistical learning theory (SLT) has received growing attention in fault detection of distributed systems [3][2]. For instance, Chen et al. [3] proposed to use the probabilistic context free grammar (PCFG) and statistical χ^2 test for detecting abnormal client request traces for system failures. The paper [2] used sub-space decomposition and a sequentially discounting EM algorithm to track frequencies of software component interactions for system failures. Those approaches extract knowledge or patterns from system normal behavior, and detect abnormal observations based on the learned knowledge. However, one shortcoming of them is their incapability of discriminating whether the detected anomaly is resulting from a real system fault or just unusual workload variations.

This paper presents a way of learning the correlations between system input \mathbf{u} and the internal observation \mathbf{x} , such as the database access frequencies. By doing so, we can reduce the false alarms caused by unusual workload changes. The rationale of our approach is based on the fact that in mission critical enterprise systems, the business logic and system functionality always impose many *implicit* contextual relationships between the input and its internal state. For instance, to accomplish a specific type of client request, some components and system resources are always activated. Once we correctly learn the dependencies between system input and internal behavior, the input variables can be utilized as a 'teacher' to monitor the observations of system state. If a fault happens, the system behavior is likely to be different from that expected from the input. By detecting these discrepancies, we can capture the system failure. In this paper we choose the database access frequencies as system observation \mathbf{x} . Each variable in \mathbf{x} represents the number of accesses of a specific database table within certain time interval. The input vector \mathbf{u} represents the system load

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'06 April 23-27, 2006, Dijon, France

Copyright 2006 ACM 1-59593-108-2/06/0004 ...\$5.00.

in which each variable denotes the number of a specific type of client request issued within the time interval. Note that the system input and internal variables are not restricted to those defined in the paper. They can be other kinds of system observations as well.

Modeling the relationship between the system input and internal status has been well studied in the modern control theory. One of the most famous methods is the state-space approach, which treats the whole system as a multiple inputs and multiple outputs (MIMO) model based on its physical properties. However, the distributed computing system has no physically plausible model. More over, *not all the input variables* contribute to the activities of system states. We have to extract the relevant subset of variables. Another challenge is that the relationship between input and state is not deterministic in distributed systems. With respect to the database accesses, some types of client requests may or may not visit the database tables depending on the parameters in the request. The mechanism of connection pooling [8] in most enterprise systems further increases the uncertainty of dependencies.

As a result of these issues, we propose to apply a statistical approach, canonical correlation analysis (CCA), to learn the *probabilistic* dependencies between system inputs and database access frequencies. By means of canonical correlation analysis, the dependence between \mathbf{u} and \mathbf{x} is encapsulated in a number of variable pairs $\{\tilde{u}_i, \tilde{x}_i\}$ with the canonical correlation $\rho_i = \text{corr}(\tilde{u}_i, \tilde{x}_i)$. Based on the magnitude of ρ_i , the system variables \mathbf{x} are transformed and divided into two subsets, $\tilde{\mathbf{x}}^{(1)}$ and $\tilde{\mathbf{x}}^{(2)}$. Each variable in the subset $\tilde{\mathbf{x}}^{(1)}$ has a highly correlated partner derived from the input \mathbf{u} . A way of supervised monitoring is put forward in the paper to check the status of variables in that subspace. The variables in the subset $\tilde{\mathbf{x}}^{(2)}$ represent the less correlated and non-relevant information in \mathbf{x} with respect to the input \mathbf{u} . Those variables are monitored in an unsupervised fashion since they can not find a ‘teacher’ from \mathbf{u} . By combining the supervised and unsupervised monitoring, we capture the activities of both subspaces of \mathbf{x} .

Our approach has been tested on a real e-commerce application hosted on the J2EE multi-tiered architecture. We modify some codes in EJB components to simulate a variety of system faults which frequently appear in real situations. Experimental results demonstrate that the CCA based detector can capture almost all those faults we injected, even in the case when the impacts of injected faults are relatively weak.

2. CORRELATION ANALYSIS

Canonical correlation analysis (CCA) was introduced in [5] for studying the relationship between two sets of variables, one for $\mathbf{u} \in R^q$ and the other for $\mathbf{x} \in R^p$. It is known that even if there is a very strong linear relationship between two sets of multidimensional variables, depending on the coordinate system used, this relationship might not be visible as a correlation. Canonical correlation analysis transforms both set of variables into pairs $(\tilde{u}_i, \tilde{x}_i)$, as shown in Figure 1, where $i = 1, 2, \dots, m$ and $m = \min(p, q)$, such that the \tilde{u}_i s are uncorrelated as are the \tilde{x}_i s, and the canonical correlations $\rho_i = \text{corr}(\tilde{u}_i, \tilde{x}_i)$ are descending, $\rho_1 \geq \rho_2 \geq \dots \geq \rho_m$. By doing so, we encapsulate the dependencies between \mathbf{x} and \mathbf{u} into a subset of variable pairs based on the values of canonical correlations.

The main part of CCA calculation is to find the transforming vectors $\mathbf{w}_{u(i)}$ and $\mathbf{w}_{x(i)}$ that maximize the correlation between two variables $\tilde{u}_i = \mathbf{w}_{u(i)}^\top \mathbf{u}$ and $\tilde{x}_i = \mathbf{w}_{x(i)}^\top \mathbf{x}$

$$\rho_i = \frac{E(\tilde{u}_i \tilde{x}_i)}{\sqrt{E(\tilde{u}_i^2)E(\tilde{x}_i^2)}} = \frac{\mathbf{w}_{u(i)}^\top C_{ux} \mathbf{w}_{x(i)}}{\sqrt{\mathbf{w}_{u(i)}^\top C_{uu} \mathbf{w}_{u(i)} \mathbf{w}_{x(i)}^\top C_{xx} \mathbf{w}_{x(i)}}} \quad (1)$$

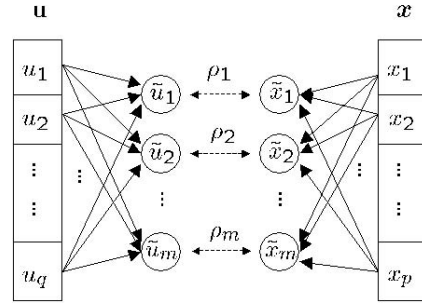


Figure 1: Canonical Correlation Analysis (CCA).

under the condition that \tilde{u}_i is uncorrelated with previous values $\tilde{u}_{i-1}, \tilde{u}_{i-2}, \dots, \tilde{u}_1$, and the same for \tilde{x}_i . In the above equation, C_{uu} and C_{xx} denote the within-set-covariance matrices of \mathbf{u} and \mathbf{x} respectively, and $C_{ux} = C_{xu}^\top$ is the between-sets-covariance matrix.

Let us look at the case where only one pair of basis vectors is sought, namely the ones corresponding to the largest canonical correlation. For simplicity, we use \mathbf{w}_u or \mathbf{w}_x to denote that pair of vectors. Since the solution of (1) is not affected by rescaling \mathbf{w}_u or \mathbf{w}_x , the problem formulated in equation (1) is equivalent to maximizing the numerator subject to

$$\mathbf{w}_u^\top C_{uu} \mathbf{w}_u = 1 \quad (2)$$

$$\mathbf{w}_x^\top C_{xx} \mathbf{w}_x = 1 \quad (3)$$

The corresponding Lagrangian is

$$L(\lambda, \mathbf{w}_u, \mathbf{w}_x) = \mathbf{w}_u^\top C_{ux} \mathbf{w}_x - \frac{\lambda_u}{2} (\mathbf{w}_u^\top C_{uu} \mathbf{w}_u - 1) - \frac{\lambda_x}{2} (\mathbf{w}_x^\top C_{xx} \mathbf{w}_x - 1) \quad (4)$$

Taking derivatives with respect to \mathbf{w}_u and \mathbf{w}_x , we obtain

$$\frac{\partial L}{\partial \mathbf{w}_u} = C_{ux} \mathbf{w}_x - \lambda_u C_{uu} \mathbf{w}_u = 0 \quad (5)$$

$$\frac{\partial L}{\partial \mathbf{w}_x} = C_{xu} \mathbf{w}_u - \lambda_x C_{xx} \mathbf{w}_x = 0 \quad (6)$$

Subtracting $\mathbf{w}_x^\top \times (6)$ from $\mathbf{w}_u^\top \times (5)$ we have

$$\mathbf{w}_u^\top C_{ux} \mathbf{w}_x - \lambda_u \mathbf{w}_u^\top C_{uu} \mathbf{w}_u - \mathbf{w}_x^\top C_{xu} \mathbf{w}_u + \lambda_x \mathbf{w}_x^\top C_{xx} \mathbf{w}_x = \lambda_x \mathbf{w}_x^\top C_{xx} \mathbf{w}_x - \lambda_u \mathbf{w}_u^\top C_{uu} \mathbf{w}_u = 0 \quad (7)$$

Together with constraint (2)(3) we conclude $\lambda_u = \lambda_x = \rho$. When C_{uu} is invertible, we get from (5)

$$\mathbf{w}_u = \frac{C_{uu}^{-1} C_{ux} \mathbf{w}_x}{\rho} \quad (8)$$

Substituting in equation (6) gives after rearranging

$$(C_{xu} C_{uu}^{-1} C_{ux} - \rho^2 C_{xx}) \mathbf{w}_x = 0 \quad (9)$$

In an analogous way we can get the equation for vector \mathbf{w}_u as

$$(C_{ux} C_{xx}^{-1} C_{xu} - \rho^2 C_{uu}) \mathbf{w}_u = 0 \quad (10)$$

Hence \mathbf{w}_u and \mathbf{w}_x are found by solving the generalized eigen problem [4] of (9) and (10) respectively. They correspond to the eigen vectors of (9) and (10) with respect to the largest eigen value. Having extracted the first pair of transforming vectors, the next canonical pairs are found in a similar way. It is shown in [5] that those solutions correspond to the eigen vectors of the same equations (9) and (10) but with different eigen values.

3. CCA BASED FAULT DETECTION

Given two sets of variables $\mathbf{u} \in R^q$ and $\mathbf{x} \in R^p$ where \mathbf{u} is the system input and represents the number of different types of client request issued within certain time interval. The vector \mathbf{x} corresponds to the access frequencies of different database tables. One option of fault detection is to track the internal variables \mathbf{x} along time and identify the abnormal behavior of \mathbf{x} with respect to its activities we already observed. However, merely concentrating on the variable set \mathbf{x} itself for detection is not robust since some anomalies of \mathbf{x} may not result from real faults but from other factors such as the unusual workload changes. The purpose of using CCA is to make use of the system input \mathbf{u} as a ‘teacher’ to provide a baseline for the activities of variables in \mathbf{x} . It can remove the uncertainties of distribution \mathbf{x} which are caused by system input. In view of information theory, our purpose is to reduce the entropy of observations by means of considering the system input, since we believe the mutual information between the status variables \mathbf{x} and input \mathbf{u} is high.

As described in the previous section, CCA transforms the two sets of variables \mathbf{u} and \mathbf{x} into m pairs $\{\tilde{u}_i, \tilde{x}_i\}$, $i = 1, 2, \dots, m$, with decreasing correlations, $\rho_1 \geq \rho_2 \geq \dots \geq \rho_m$. Based on the value of ρ_i , we decompose the space \mathbf{x} into two subsets $\tilde{\mathbf{x}}^{(1)}$ and $\tilde{\mathbf{x}}^{(2)}$. Each variable \tilde{x}_i in subset $\tilde{\mathbf{x}}^{(1)}$ has a partner \tilde{u}_i from input \mathbf{u} that is highly correlated ($\rho_i \geq \rho^*$, here we choose the threshold $\rho^* = 0.9$). The variables in $\tilde{\mathbf{x}}^{(2)}$ represent the low correlated and uncorrelated part of \mathbf{x} . Then we introduce two different strategies to monitor the variables in $\tilde{\mathbf{x}}^{(1)}$ and $\tilde{\mathbf{x}}^{(2)}$ respectively.

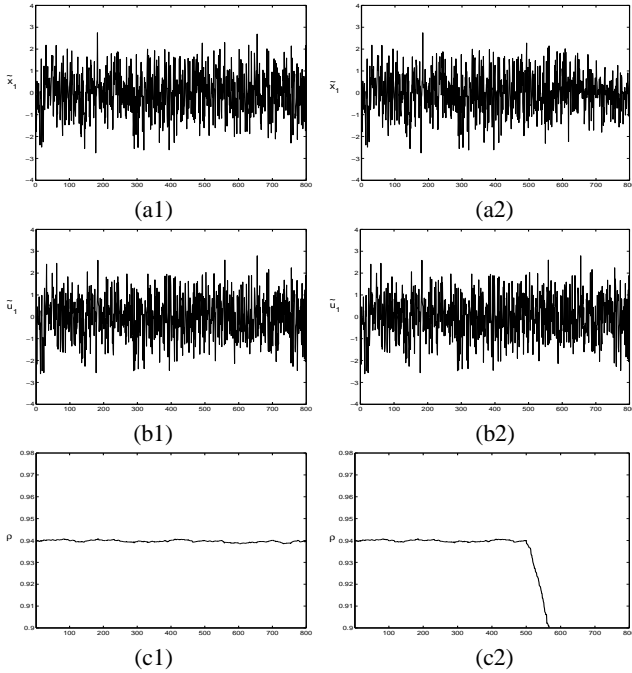


Figure 2: Supervised monitoring. (a1)(a2) the state variable \tilde{x}_1 in normal and faulty cases; (b1)(b2) the input variable \tilde{u}_1 in normal and faulty cases; (c1)(c2) the correlation ρ between \tilde{u}_1 and \tilde{x}_1 in normal and faulty cases.

We propose supervised monitoring for each variable \tilde{x}_i in $\tilde{\mathbf{x}}^{(1)}$. Its partner \tilde{u}_i serves as a teacher to monitor the behavior of \tilde{x}_i . Figure 2 demonstrates the process of this supervised monitoring. We plot the values of \tilde{x}_1 in system normal status (Figure 2(a1)) and faulty status (Figure 2(a2)) respectively. It is hard to detect the

system fault just based on \tilde{x}_1 itself because the values of \tilde{x}_1 are very diverse. However, once we have the knowledge of its highly correlated partner \tilde{u}_1 , as shown in Figure 2(b1)(b2), the correlation between \tilde{u}_1 and \tilde{x}_1 can be calculated and updated. Figure 2(c1)(c2) illustrate the correlation curves in normal and faulty cases respectively. It is easy to see that in faulty case, the correlation between the signal \tilde{x}_i and \tilde{u}_i drops after the 500th observation because the system encountered some abnormal failures. Note the horizontal axis in all these figures represents the time dimension.

To implement this supervised detector, we need to : 1) obtain the projection vector $w_{u(i)}$ and $w_{x(i)}$ for each pair $\{\tilde{u}_i, \tilde{x}_i\}$ and their correlation ρ_i ; 2) find ways of updating the correlation ρ_i for each new observation; 3) determine the threshold for each ρ_i that represents its deviation from normality. To accomplish this, we collect observations of \mathbf{x} and \mathbf{u} during system normal operations as the training data, and split them into two parts. The first data set is used to extract the correlation model, e.g. the projection vectors and canonical correlation for each pair $\{\tilde{u}_i, \tilde{x}_i\}$, between \mathbf{x} and \mathbf{u} by using CCA. The second training set is used to determine the threshold for each ρ_i . Starting from the previous learned CCA model, we sequentially update the correlation ρ_i for every observation in the second data set. Given k th observation \mathbf{x}^k and \mathbf{u}^k , an *exponentially weighted moving average* (EWMA) filter is employed to update the within-set-covariance matrices, C_{xx} and C_{uu} , and the between-sets-covariance matrix C_{xu} . For instance, the EWMA based updating of between-set-covariance matrix is expressed as

$$C_{xu}^{k+1} = \gamma C_{xu}^k + (1 - \gamma) \mathbf{x}^k (\mathbf{u}^k)^\top \quad (11)$$

where the constant γ dictates the degree of filtering. When we choose $\gamma = \frac{1}{k+1}$, equation (11) changes into the traditional *moving average* (MA) estimation. In the EWMA filter, the parameter γ is fixed so that C_{xu}^{k+1} can ‘age out’ old observations and put more importance to the recent data. This allows the algorithm to automatically adapt to the system changes. In the paper we choose $\gamma = 0.99$. The two within-sets-covariance matrices are updated in the similar way. Note here it is assumed that \mathbf{x} and \mathbf{u} are zero-mean variables. If not, we can easily center them by subtracting them from the mean obtained from the first set of training data. Once we obtain all the values of ρ_i , its mean and standard deviation are calculated. The threshold is then determined as 3 times standard deviation below the mean. During the online monitoring process, the correlations ρ_i s are updated in the same way. Whenever some of the updated correlations fall below their thresholds, we conclude that the system has faulty behavior.

Since the variables in the subset $\tilde{\mathbf{x}}^{(2)}$ can not find a highly correlated partner from the input \mathbf{u} , they are monitored in an unsupervised manner. There are a variety of methods for unsupervised monitoring in the literature [7][2][6]. Here we define a statistic s to capture the activities of variables in $\tilde{\mathbf{x}}^{(2)}$

$$s = \sum_{i=1}^{m_2} \tilde{x}_i^2 \quad (12)$$

where m_2 is number of variables in $\tilde{\mathbf{x}}^{(2)}$, and \tilde{x}_i is a zero-mean variable with unit standard deviation according to equation (3). If we assume that \tilde{x}_i s are normally distributed, then s obeys the χ^2 distribution with degree of freedom m_2 . The threshold for s is then determined by choosing certain confidence level of χ^2 distribution. In the experiments, we choose confidence level $p = 0.999$ to decide the threshold. The geometric interpretation of equation (12) is that the score s actually represents the distance between the projection of \mathbf{x} into the subspace spanned by $\tilde{\mathbf{x}}^{(2)}$ and the origin of that subspace.

4. EXPERIMENTAL RESULTS

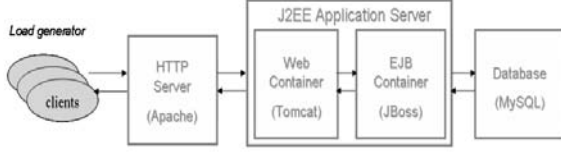


Figure 3: The architecture of test bed system.

Our algorithm has been tested on a real e-commerce application which is based on J2EE multi-tiered architecture. J2EE is a widely adopted platform standard for constructing enterprise applications based on deployable java components, called Enterprise Java Beans(EJBs). The architecture of system under test is shown in Figure 3. We use Apache as a web server. The application server consists of the web container (Tomcat) and the EJB container (JBoss). The MySQL is running at the back end to provide persistent storage of data. PetStore 1.3.2 is deployed as our test bed application. Its functionality consists of store front, shopping cart, purchase tracking and so on. There are 47 components in PetStore, including EJBs, Servlets and JSPs. We built a client emulator to generate a workload similar to that created by typical user behavior. The emulator produces a varying number of concurrent client connections with each client simulating a session based on some common scenarios, which consists of a series of requests such as creating new accounts, searching by keywords, browsing for item details, updating user profiles, placing order and checking out. Our experiments are conducted under these simulated workloads.

The CCA is applied to model the contextual relationship between system load and database access frequencies. From the Apache server log, there are 12 different client HTTP request types issued in PetStore. As a result, the input vector $\hat{\mathbf{u}}_t$ is defined as consisting of 12 variables. Each variable in $\hat{\mathbf{u}}_t$ represents the number of specific type of client request issued within certain time interval Δt , observed at time t . Here we choose $\Delta t = 10s$. Similarly we find out 6 independent database tables from MySQL database log. The vector $\hat{\mathbf{x}}_t$ is hence defined to the number of accesses for each database table within Δt . Considering the time delays for transmitting client requests, we define $\mathbf{u} = [\hat{\mathbf{u}}_t \ \hat{\mathbf{x}}_{t-\Delta t} \ \hat{\mathbf{u}}_{t-\Delta t}]$, $\mathbf{x} = \hat{\mathbf{x}}_t$ to account for the effect caused by the delay.

The training data are collected under system normal operations and divided into two parts. The first part of observations is used to calculate the correlation parameters, such as vectors $\mathbf{w}_{u(i)}$, $\mathbf{w}_{x(i)}$, and ρ_i , $i = 1, 2, \dots, m$, where $m (= 6)$ equals to the minimum dimension of \mathbf{x} and \mathbf{u} . The calculated values of ρ_i s are

$$\boldsymbol{\rho} = [0.999, 0.998, 0.996, 0.966, 0.759, 0.341] \quad (13)$$

Based on the magnitude of ρ_i , the original space \mathbf{x} is divided into two subspaces, $\tilde{\mathbf{x}}^{(1)}$ and $\tilde{\mathbf{x}}^{(2)}$, with dimension 4 and 2 respectively.

The remaining training data are used to sequentially update the ρ_i s and s score, and then determine their thresholds for anomaly. We sequentially calculate the covariances (11) and update each correlations ρ_i according to equation (1). The threshold for each ρ_i is determined as $m_i - \max(3\sigma_i, 0.01)$, where m_i is the mean observation of ρ_i obtained from the training set, and σ_i is its standard deviation. The function $\max(., .)$ is used to reduce the false positives caused by some training data with extremely small or zero variances. Similarly the score s in (12) is also sequentially updated and its threshold is determined based on the confidence level $p = 0.999$ for χ^2 distribution with two degrees of freedom, according to the discussion in Section 3.

Two pieces of test data sets are generated under system normal operation. Note that the workloads for generating these data are totally different from that for generating the training data. The experimental results for these data sets are shown in Figure 4, in which each figure presents the correlation and s score curves calculated by CCA method, together with the threshold for each measure plotted as a dashed line. It is shown that CCA based approach works well for normal test data. There are no false positives reported.

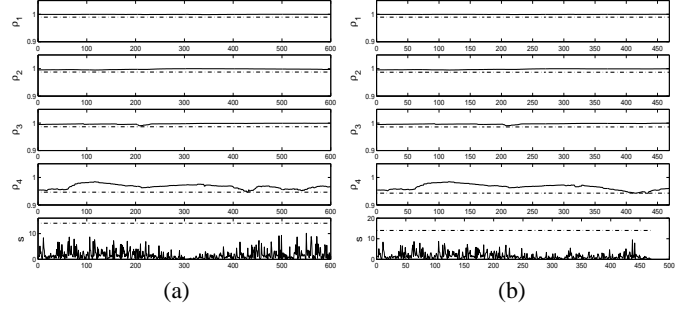


Figure 4: Canonical correlation and s score curves obtained from CCA. (a)Normal Case I; (b)Normal Case II.

We also modify the codes in some EJB components to simulate following real system errors, and test the performance of our fault detector under those failure cases.

Memory Leaking: We create a collection class object in a persistent EJB, *ShoppingCartLocalEJB*, and add an additional procedure that always allocates new objects in the collection without any intention of releasing its usages. Since the reference of the collection object is always pointed from other objects, the garbage collector does not know whether the inside of the collection is useful any more or not. Hence the PetStore application will gradually exhaust the supply of virtual memory pages, which leads to severe performance issue and makes the accomplishment of client requests much slower. As a result, the contextual correlation model learned during system normal status does not hold anymore. Our approach quickly detected such failure. As shown in Figure 5(a), the canonical correlation ρ_2 drops significantly below the threshold. In addition, other correlation scores, such as ρ_3 and ρ_4 , and the s score all exhibit deviated behaviors from the thresholds.

File Missing: In the packaging process of Java web applications, it might happen that a file is improperly dropped from the required composition which will result in failures of invoking correct system response, and may eventually cause service malfunction which makes the user come across *strange* web pages. To simulate this type of error, we dropped some JSP files in the PetStore application. Since the JSPs in the application server are usually dynamic pages, which need some data obtained from the database, the missing of those files will significantly affect access patterns of the database. Our experimental results proved this conclusion. From Figure 5(b), we see that the evidence for the fault, shown in the correlation ρ_4 , is strong enough to be detected by our approach.

Deadlock: We modify the function *updateItemQuantity()* in *ShoppingCartLocalEJB*, in which a variable is introduced to intermittently trigger the thread to sleep for a while and then recover. This process is to simulate the phenomenon of deadlock failure. Consider the case when the *ShoppingCartLocalEJB* component becomes deadlocked with other threads due to competing for the same database resources, all the functionalities of *ShoppingCartLocalEJB* will become *silent* just like the thread is in the sleep mode. However, after a while when the database deadlock management tools detect this

deadlock and release it, the component *ShoppingCartLocalEJB* becomes alive again. By tuning the frequency of the trigger, we simulate that around 5 percent of requests passing through the *ShoppingCartLocalEJB* component get locked for a time period between 2 and 4 seconds. This is an example of failure with *extremely weak* impact. However, our detector still can identify the evidence of such failure. Figure 5(c) illustrates that the s score gets slightly affected by that software bug.

Busy Loop: The actual causes of request slowdown can be quite a few, such as the spin lock fault among synchronized threads. We simulate the phenomenon of slowdown by adding a busy loop procedure in the code. Depending on the position of instrumentation, the significance of simulation is different. In this section we simulate such fault that all the client requests going through the *ShoppingCartLocalEJB* component get affected. The experimental results shown in Figure 5(d) demonstrate good performances in dealing with this fault case. The correlation and s curves show significant changes in our method.

Expected Exception: The *expected exception* fault [3] happens when a method declaring exceptions (which appears in the method’s signature) is invoked. In this situation an exception is thrown without the method’s code being executed. Even though applications are expected to mask from end user such exceptions, it is still possible in real situations that they are not handled well and then turn into run time failures. From Figure 5(e) we see that the correlation ρ_4 has been significantly affected in the expected exception failure. Note the traditional detection tools, which are based on the operational statistics such as response time, are not able to detect such fault because the expected exception does not crash the application software and the response time for delivering the client requests are still within normal thresholds.

Null Call: The *null call* fault [3] causes all methods in the affected component to return a null value without executing the methods’ code. This situation can arise at runtime from neglect of allocating certain resources, failed lookups, etc. Similar as the *expected exception*, the null call fault results in subtle outcomes, and does not cause exceptions to be printed on the operator’s console, and does not crash the application software. On the other hand, these bugs can easily happen in practice due to incomplete, or incorrect, handling of rare conditions. The detection results of null call fault, as shown in Figure 5(f), are very similar to those in the expected exception case.

5. CONCLUSIONS

In this paper we have presented a new approach for fault detection in distributed systems. We have utilized the information about system input and proposed the concept of supervised and unsupervised monitoring. The statistical approach, canonical correlation analysis (CCA), has been employed to learn the implicit correlations between two sets of variables, the system workload and database access frequencies. By doing so, the variables about the database accesses are divided into two subsets. One is highly correlated with the input, and each variable in that subset is monitored with the aid of the input. The other subset accounts for the variables is less correlated or uncorrelated with the input, which is monitored in an unsupervised way. We have also simulated a variety of faults in a real e-commerce application. By tuning the significance of injected faults, we show that our approach is able to detect failures at their early stages when the impacts are very weak.

6. REFERENCES

[1] M. K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in

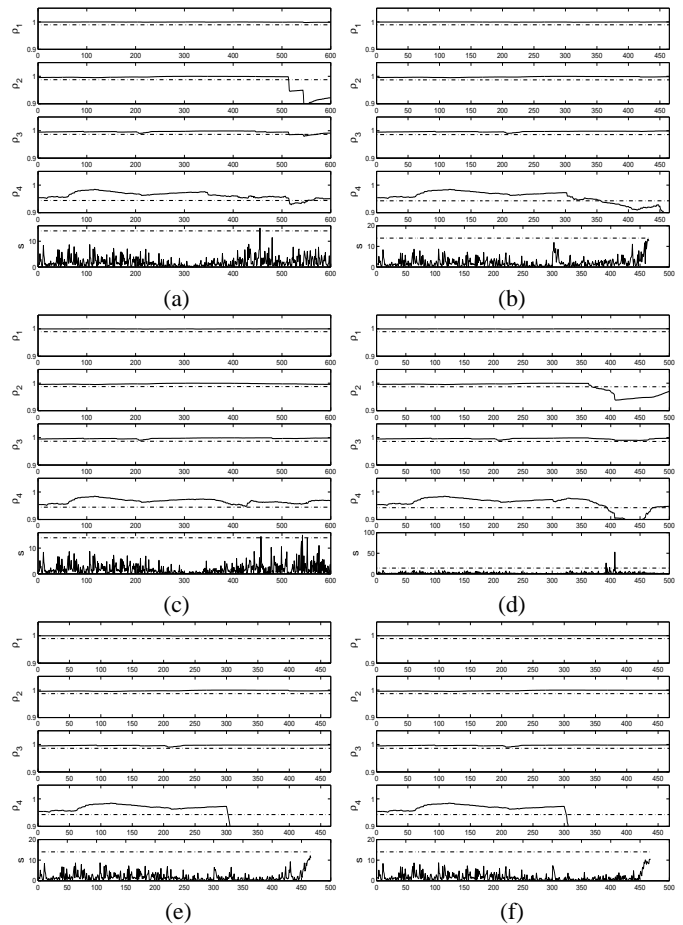


Figure 5: Canonical correlation and s score curves obtained from CCA for different types of faults. (a)Memory Leaking; (b)File Missing; (c)Deadlock; (d)Busy Loop; (e)Expected Exception; (f)Null call.

partitionable networks. *Theoretical Computer Science, special issue on distributed algorithms*, 220:3 – 30, 1999.

[2] 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 and Data Mining*, pages 750–755, Chicago, IL, August 2005.

[3] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic systems. In *2002 International Performance and Dependability Symposium*, Washington, DC, June 2002.

[4] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, third edition, 1996.

[5] H. Hotelling. Relations between two sets of variables. *Biometrika*, 28:321–377, 1936.

[6] Tsuyoshi Idé and Hisashi Kashima. Eigenspace-based anomaly detection in computer systems. In *the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 440–449, Seattle, WA, August 2004.

[7] K. Yamanishi, J. Takeuchi, G. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 320–344, 2000.

[8] Sun Microsystems. J2ee connector architecture specification, version 1.0 (public draft). 2000.