We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

17

Anomaly-based Fault Detection with Interaction Analysis Using State Interface

Byoung Uk Kim
The University of Arizona
USA

1. Introduction

Fault detection, analysis, and recovery with effective monitoring in distributed systems is a challenging research problem due to the exponential growth in scale and complexity of system resources and applications, the continuous changes in software and hardware configurations, and the heterogeneous services being offered and deployed. In spite of enormous advances in hardware and software technology, there are many uncertainties and unpredictable operations in distributed systems that could be triggered by one or a combination of several events such as network failures, intermittent software failures, bugs in software and services, etc. Various distributed systems especially employed in safety-critical environments must work correctly in spite of the occurrence of faults.

The fault detection and analysis approach presented in this chapter is based on hardware/software fault tolerance techniques and data mining techniques including regression trees, neural networks, multivariate linear regression, fuzzy classification, logistic regression, classification tree, naïve bayes , and sequential minimal optimization.

In this chapter, we present an innovative approach to detect faults (hardware or software) and also identify the source of the faults. Our online monitoring mechanism collects significant interactions among system state components such as CPU, memory, I/O, and network interface in real-time between all the components of a distributed system. We record and trace these runtime properties and analyze all the interactions using data mining and supervised learning techniques to acquire the rules that can accurately model the normal interactions among these components. We have implemented an anomaly-based fault detection engine and used it to detect faults in a typical multi-tier web based ecommerce environment that implements ecommerce transactions based on the TPC-W web ecommerce benchmark (TPC-W, 2005).

The organization of the remaining sections of the chapter is as follows. In section 2, we review related work. In section 3, we explain theoretical framework including system presentation, normal and abnormal definition with system interfaces and attribute definitions. In section 4, we present anomaly analysis methodology to implement efficient fault detection and analysis. We discuss data sources, training and testing data and fault types used in our experiments and then present our experimental results and evaluation of our approach in section 5 and 6. In section 7, we summarize the chapter and discuss future research activities.

Source: Theory and Novel Applications of Machine Learning, Book edited by: Meng Joo Er and Yi Zhou,
ISBN 978-3-902613-55-4, pp. 376, February 2009, I-Tech, Vienna, Austria

2. Related work

Fault detection and analysis has always been an active research area due to its importance to distributed systems and their applications. It is necessary for the system to be able to detect the faults and take the appropriate actions to avoid further degradations of the service. In this section, we classify fault detection techniques based on different categories such as hardware and software techniques and based on the detection schemes such as statistical methods, distance and model based methods and profiling methods.

2.1 Hardware based

(Reinhardt & Mukherjee, 2000) proposed Simultaneous and Redundant Threading (SRT) to provide transient fault coverage with high performance by taking advantage of the multiple hardware contexts of Simultaneous Multithreading (SMT). It provides high performance by using active scheduling of its hardware components among the redundant replicas and reduces the overhead of validation by eliminating cache misses. (Ray et al., 2001) proposed modern superscalar out-of-order datapath by modifying a superscalar processor's micro-architectural components and validating the redundant outcomes of actively duplicated threads of execution, while the fault recovery plan uses the branch-rewind mechanism to restart at a place which error happened. Commercial fault-tolerant systems combine several techniques such as error correcting codes, parity bits and replicated hardware. For example, Compaq Non-Stop Himalaya (Wood, 2004) employs '*lockstepping*' which runs the same program on two processors and compares the results by a checker circuit.

2.2 Software based

(Reis et al., Dec. 2005) introduced PROFiT technique regulating the stage of reliability at fine granularities by using software control. This profile-guided fault tolerance determines the weakness and performance trade-offs for each program region and decide where to turn on and off redundant by using a program profile. (Oh et al., 2002) proposed Error Detection by Duplicated Instructions (EDDI) which copies all instructions and inserts check instructions for validation. Software based mechanisms present high reliability gain at low hardware cost and high fault coverage. However the performance degradation and failure to directly check micro-architectural components result in another trend of fault detection, hybrid redundancy techniques (Reis et al., June. 2005) such as CompileR-Assisted Fault Tolerance (CRAFT).

We classify detection and analysis strategy based on the following approaches such as statistical, profiling, model-based, and distance-based methods.

2.3 Statistical methods

This method traces the system behavior or user activity by gauging variables over time such as event message between components, system resource consumption, login/out time of each session. It maintains averages of these variables and detects the anomaly behavior by making a decision whether thresholds are exceeded based on the standard deviation of the variables monitored. It also compares profiles of short/long term user activities using complex statistical models. (Ye & Chen, 2001) employs chi-square statistics to detect anomalies. In this approach, the activities on a system are monitored through a stream of events and they are distinguished by event type. For each event type, the normal data from

audit events are categorized and then used to get chi-square for difference between the normal data and testing data. It considers large deviations as abnormal data.

2.4 Distance based methods

One of the limitations of statistical approaches is that it becomes inaccurate and hard to calculate approximately the multidimensional distributions of the data points when outlier detection exists in higher dimensional spaces (Lazarevic et al., 2006). Distance based methods try to overcome this limitation and identify outliers by computing distances among points. For example, (Cohen et al., 2005) presents an approach using usual clustering algorithms such as k-mean and k-median to get the system status. The difference with others focusing on clustering algorithm is that they use a signature and show the efficacy for clustering and connection based recovery by means of distinguished techniques such as pattern recognition and information retrieval.

2.5 Model based methods

This method describes the normal activity of the monitored system by using different types of models and identifies anomalies as divergence for the model that characterizes the normal activity. For example, (Maxion & Tan, 2002) obtains a sequential data streams from a monitored procedure and employs a Markov models to decide whether the states are normal or abnormal. It calculates the probabilities of transitions between the states using the training data set, and utilizes these probabilities to evaluate the transitions between states in test data set.

2.6 Profiling methods

It builds profiles of normal behavior for diverse types of systems, users, applications etc., and variations from them are regarded as anomalous behaviors. These profiling methods vary significantly different data mining techniques while others use various heuristic based approaches. In data mining methods, each case in training data set is configured as normal or abnormal and a data mining learning algorithm is trained over the configured data set. By using these methods, new kinds of anomalities can be detected in fault detection models with retraining (Lazarevic et al., 2006). (Lane & Brodley, 1999) uses a temporal sequence learning technique in profiling Unix user commands for normal and abnormal scenarios. It then uses these profiles to detect any anomalous user activity. Other algorithms for fault detection include regression trees, multivariate linear regression, logistic regression, fuzzy classification, neural networks and decision trees.

3. Theoretical framework

3.1 System presentation

Consider a general *n* input *m* output nonlinear dynamic system which can be expressed by the Nonlinear Auto Regressive Moving Average (NARMA) model (Chenand & Billings, 1994) as

$$
\begin{aligned}
y_{\hat{\lambda}}(k) &= f_{\hat{\lambda}}(\bar{y}, \bar{\mathrm{u}}, \theta) \\
\bar{y} &= \{y(k-1), y(k-2), \ldots, y(k-n)\} \\
\bar{\mathrm{u}} &= \{u(k-\gamma-1), u(k-\gamma-2), \ldots, u(k-\gamma-m)\}
\end{aligned} \tag{1}
$$

where $f:\Re^P \times \Re^Q \to \Re,$ with $P=\sum_{i=1}^{m} p_i$, $Q=\sum_{j=1}^{n} q_j$ is the mathematical realization of the system dynamics for the λ_{th} output. $y(k)\in\Re$ is the output of the system at sample instant k, $u(k)\in\Re$ is the input to the system. p and q are the lengths of the regression vectors of $\overline{y}$ and $\overline{u}$, respectively. f is a recognized nonlinear function describing the dynamic characteristics of the system. γ is the relative degree of the system. n and m are known system structure orders. θ is the system parameter vector whose unanticipated changes are regarded as faults in the system. It represents all the potential faults in the nonlinear system such as sensors and processes.

In our research, we use available system input and output, respectively $\overline{u}$ and $\overline{y}$ to detect and predict any undesirable changes in θ. To simplify the presentation, we assume that during the initial stage $k\in[0,T]$, the normal and healthy values of these values are available and can be obtained from the system under consideration. There is a certain normal trajectory related with system interfaces. So, we can say the system parameter vector in normal activities is θ_N. It means that there is a known θ_N such that

$$\theta = \theta_N \tag{2}$$

To define the system abnormality, we may construct a redundant relation such that

$$m_{\lambda}(k) = | y_{\lambda}(k) - f_{\lambda}(\overline{y}, \overline{u}, \theta_N) | \tag{3}$$

If $m(k)$ is large enough by checking the values against a pre-specified threshold, we say there is abnormality in the system. Contrarily, if $m(k)$ is very small, the system is normal.

3.2 Normal and abnormal definition with system interfaces

We develop a systematic framework for identifying potential system abnormality. Ψ denote the system attributes, and $\Psi^{\Diamond}$ is the set of sequences over the alphabet Ψ. We say that system attributes $S\in\Psi^{\Diamond}$ is accepted by the detection system if executing the sequence S = {S_1, S_2, S_3, ….. } does not trigger any alarm. Let $N\subseteq\Psi^{\Diamond}$ denote the set of system attributes allowed by the detection system, i.e.,

$$N \overset{def}{=\!=} \left\{ S\in\Psi^{\Diamond} : \text{S is accepted by the detection system} \right\} \tag{4}$$

Also, let $A\subseteq\Psi^{\Diamond}$ denote the set of system attributes that are not allowed by the detection system, i.e.,

$$A \overset{def}{=\!=} \left\{ S\in\Psi^{\Diamond} : \begin{array}{c}\text{S is an equivalent variant} \\ \text{on the given suspicious sequence}\end{array} \right\} \tag{5}$$

Now we can state the conditions for the existence of abnormality in distributed systems. The set $N \cap A$ is exactly the set of system attributes that give the suspicious or abnormal status to host without detection, and thus the abnormalities are possible if $N \cap A \neq \varnothing$.

3.3 Attribute definitions

We use the following attributes for the abnormality analysis: Attribute rate (AR) per target attribute, Component rate (CR) per target component, Aggregate System rate (ASR) per target system and Number of abnormal session (NAS)

Definition 1

AR per target address is used to find out the current flow rate for a given target IP address P_j as observed by a interface monitor and can be computed as in (1)

$$AR\ (A_i,\ P_j,\ t)\ =\ \{\sum_{t=T}^{2T} DT(t)\}\Big/\mathrm{T} \tag{6}$$

where DT denotes the number of data belonging to attribute A_i and to a target P_j within a given time T.

Definition 2

CR per target address is used to determine the current flow rate as observed by a Interface monitor for all the attributes A_i that go through the same interface (I_k) and have the same target IP address (P_j). This metric can be computed as in (2)

$$CR\ (I_k,\ P_j,\ t)\ =\ \sum\nolimits_{\forall\ i} AR\ (A_i,\ P_j,\ t) \tag{7}$$

Definition 3

ASR per target address denotes the current flow rate for a given target IP address P_j as observed by Interface monitor

$$ASR\ (P_j,\ t)\ =\ \sum\nolimits_{\forall\ k} CR\ (I_k,\ P_j,\ t) \tag{8}$$

Definition 4

NAS the number of abnormal sessions for a target P_j as observed by Interface monitor

$$NAS\ (P_j,\ t)\ =\ \sum\nolimits_{\forall\ i} (1-S_i) \tag{9}$$

where S_i is a binary variable that is equal to 1 when the session is successful and 0 and when it is not.

4. Abnormality analysis methodology

Our approach is based on autonomic computing paradigm that requires continuous monitoring and analysis of the system state, and then plan and execute the appropriate actions if it is determined that the system is deviating significantly from its expected normal behaviors as shown in Figure 1.

By monitoring the system state, we collect measurement attributes about the CPU, IO, memory, operating system, and network operations. The analysis of this data can reveal any anomalous behavior that might be triggered by failures.

Once a fault is detected, the next step is to identify the appropriate fault recovery strategy to bring the system back into a fault-free state. In this paper, we focus on monitoring and analyzing the interactions among these components to detect any hardware or software failures.

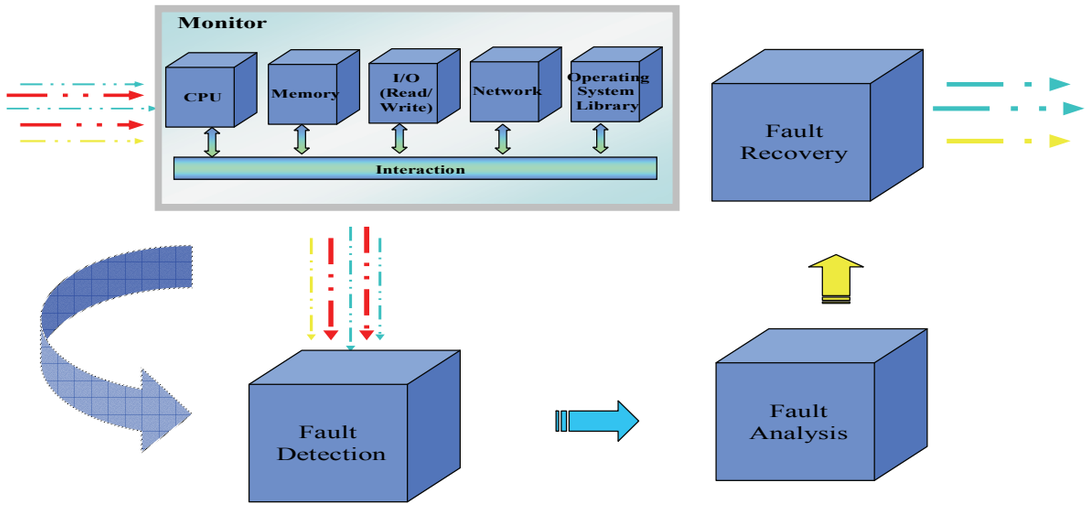


Fig. 1. Self-healing Engine

4.1 Monitoring and analysis of system component operations

The first step is to identify a set of measurement attributes that can be used to define the normal behaviors of these components as well as their interactions with other components within the distributed system. For example, when a user runs a QuickTime application, one can observe certain well defined CPU, Memory and I/O behaviors. These operations will be significantly different when the application experience un-expected failure that leads to application termination; one can observe that the application, although consuming CPU and memory resources, does not interact normally with its I/O components.

These monitored data are analyzed using two types of vector-based (VR) metrics (Wood, 2004): Local-VR (LVR) and Global-VR (GVR). The LVR is used to evaluate the interface interaction, detection, analysis and recovery measurements associated with a specific fault, *f*, in the set of faults in the fault hypothesis, *F* as shown in Equations 10-13:

$$\overrightarrow{LVR}_{\text{interfaces } f_i \in F} = (\text{CPU,MEM,I/O,NET,OS}) \tag{10}$$

$$\overrightarrow{LVR}_{\text{detection } f_i \in F} = (\text{AR, CR, ASR, NAS, Precision, Recall, F-measure, False negative rate, Fals positive rate, Time to detect, ...}) \tag{11}$$

$$\overrightarrow{LVR}_{\text{analysis } f_i \in F} = (\text{Accuracy, Time to analyze, Analyzed or not?}) \tag{12}$$

$$\overrightarrow{LVR}_{\text{recovery } f_i \in F} = (\text{Accuracy, Time to recover, Recovered or not?}) \tag{13}$$

'CL' refers to metrics measured on the client, and 'DB' refers to metrics measured on the database server. Equation (10) is used to identify the measurement attributes associated with the interactions among system components. When a specific fault, e.g. application abnormal termination is generated in the system, $\overrightarrow{LVR}_{\text{interfaces } f_i \in F}$ is composed with the

related measurement attributes. For example, $\overrightarrow{LVR}_{\text{interfaces } f_i \in F}$ in abnormal termination of QuickTime application will consist of measurements of user CPU, active memory in directory and I/O read activities as shown in Equation 10. Equation 11 describes several metrics used to find out the characteristic of interfaces explained in section 3.3 and evaluate the performance of the fault detection strategy using several metrics: 1) False negative rate is the percentage of abnormal flows incorrectly classified as normal 2) Precision is the proportion of correctly detected abnormal flows in the set of all normal flows returned by detection. 3) Recall is the number of correctly detected abnormal flows in retrieved as fraction of all abnormal flows. And 4) F-measure is used to quantify the trade-off between recall and precision. All these performance metrics were explained in (Kim & Hariri, 2007). Equation 12 and 13 evaluate the accuracy of our approach using several metrics such as: 1) Accuracy is used to estimate whether all actual faults are correctly identified. 2) Time to analyze/recovery is used for measuring the time taken the system to analyze/recovery from faults.

The global vector based metric GVR quantifies in a similar way to LVR the quality of the analysis, detection and recovery as shown in Equations 14-16.

$$\overrightarrow{GVR}_{\text{detection}} = (\text{Throughput, Response time, Availability, Cost, Time}) \tag{14}$$

$$\overrightarrow{GVR}_{\text{analysis}} = (\text{Throughput, Response time, Availability, Cost, Time}) \tag{15}$$

$$\overrightarrow{GVR}_{\text{recovery}} = (\text{Throughput, Response time, Availability, Cost, Time}) \tag{16}$$

The main difference between LVR and GVR is in defining the target. That means LVR is used to evaluate the measurements associated with a specific fault in the set of faults but GVR is used to evaluate the role of the target system in a given environment. LVRs in each fault have an effect on the GVR. For example, if there is a memory related fault, we may see interface interactions in $\overrightarrow{LVR}_{\text{interfaces}}$ resulting in value changes in $\overrightarrow{LVR}_{\text{detection}}$. These changes also affect $\overrightarrow{GVR}_{\text{detection}}$ allowing us to evaluate the healthiness of the target system. This operation flow is depicted in Figure 2. We need this classification because our target system to detect the faults is in the domain of distributed system. Equations (14), (15), and (16) describe the metrics to measure the performance of the target system in detection, analysis, and recovery: 1) Throughput is the summation of data rates sent to all system nodes. 2) Response time means the time taken to react to a given goal such as 'detect' and 'recover' in the system. 3) Availability is the ratio of the total time through which the system is available to the overall total time.

Normal execution will have a certain trajectory with respect to system interfaces as shown in Figure 2. For example, if the disk is behaving normal without any fault, this can be recognized by obvious disk trajectory over time. In case of abnormal state, all information monitored in the normal state will have different trajectories. We capture and monitor this trajectory features, train this interface trajectory with $\overrightarrow{LVR}$ and $\overrightarrow{GVR}$ by using rules generated from the training data set and apply them at runtime. It shows normal trajectory drifting to suspicious trajectory by exemplifying one of $\overrightarrow{LVR}_{\text{interfaces } f_i \in F}$ and one of

Metric	Description
var_CL_CPUIF _KERNAL	variance of the number of kernel time spent on client received through cpu interface
var_CL_CPUIF _USER	variance of the number of user time spent on client received through cpu interface
val_CL_MEMIF_ACT IVE	value of number of memory hat has been used more recently and usually not reclaimed unless absolutely necessary received through memory interface on client
val_CL_MEMIF_IN_ DIR	value of number of memory that takes more work to free on client
val_CL_MEMIF_IN_ TARGET	value of number of memory that kernel uses for making sure there are enough inactive pages around on client
val_CL_IOIF _READ	value of IO load read received through IO interface on client
val_CL_IOIF _WRITE	value of IO load read received through IO interface on client
var_DB_IOIF _READ	value of IO load read received through IO interface on database server
val_CL_CON _SWITCHES	value of the number of context switches that counts the number of times one process was ``put to sleep'' and another was ``awakened'' on client

Table 1. A sample of metrics used to characterize the interaction among components

$\overrightarrow{GVR}_{\text{detection}}$ metrics over time. Rules can be formed of the interface metrics including states, events, state variables and time of transitions. These rules are generated to evaluate system healthiness as following:

Rule 1: $M_i \le I_t \le M_a \quad \mathrm{T} \in (t_j, t_j + k)$

Interface I_t in some time k starting at time t_j is delimited by high value M_a and low value M_i when defined normal flow occurs.

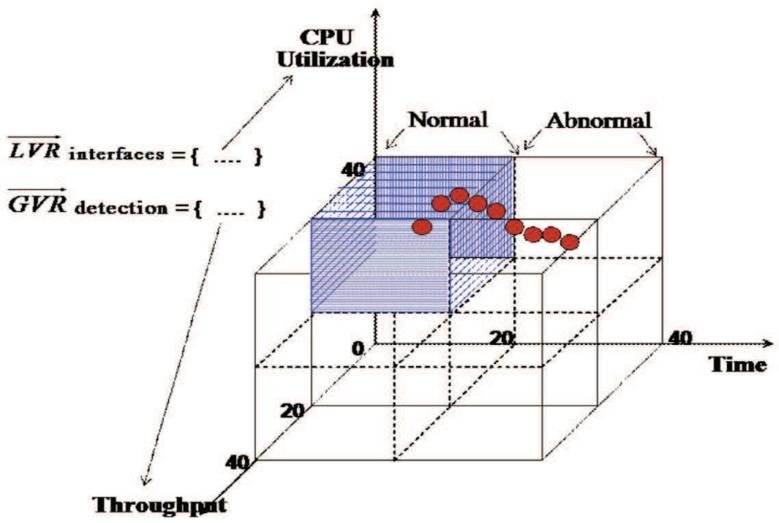


Fig. 2. Abnormality identification: LVR and GVR drift

Rule 2: $M_i \le I_n \le M_a \ \forall T \in (t_j, t_j + k) \Rightarrow M_i' \le I_a \le M_a' \ \forall T \in (t_g, t_g + r)$

Interface I_n delimited by high value M_a and low value M_i in time k will affect another interface I_a delimited by high value M_a' and low value M_i' and it will last some period r. This rule is about interaction among interfaces and it needs higher latency than the previous rule because of matching more interfaces

We build a fault detection prototype to demonstrate the utility of VR metrics to detect faults. Figure 3 shows a sampling of metrics among more than 40 interfaces and real data monitored at the client side. For clarity, we pick a sample of metrics as shown in Table 1. All data monitored in the experiment with fault injections are stored in a database and the rules are generated during the training stage. In Figure 3, the set of data from time t_0 to the time before t_i is normal with respect to a given workload. The first injected fault at t_i shows the interaction with 5 interfaces such as var_CL_CPUIF_KERNAL, val_CL_MEMIF_ACTIVE, val_CL_MEMIF_IN_DIR, var_CL_CPUIF_USER, and val_CL_MEMIF_IN_TARGET.

	var_CL_CPUIF_KERNAL	var_CL_CPUIF_USER	val_CL_MEMIF_ACTIVE	val_CL_MEMIF_IN_DIR	val_CL_MEMIF_IN_TARGET	val_CL_IOIF_READ	val_CL_IOIF_WRITE
Time t_0	5.47264	16.418	135600	23	30612	398	654
	4.47761	35.323	135908	135	30608	423	234
	6.49751	10.995	134316	123	30612	478	432
	7.49254	20.995	134316	98	30612	222	311
	4.99503	25.995	134312	209	30612	209	309
	13.49751	18.995	134312	245	30612	390	512
	11.99503	31.493	134316	154	30612	312	265
Fault t_i	16.44779	83.582	12[illegible]4	15[illegible]6	3[illegible]4	243	398
	18.4408	81.582	12[illegible]2	10[illegible]0	3[illegible]8	354	387
t_j	9.95025	5.4726	134796	58	30608	335	254
	4.47761	5.4726	134556	0	30612	512	401
Fault t_k	14.92254	38.875	12[illegible]8	10[illegible]0	3[illegible]4	345	299
	17.9104	4[illegible]9	12[illegible]6	15[illegible]2	3[illegible]0	478	289
t_l	22.[illegible]07	15[illegible]26	133628	579	30612	391	312
	21.[illegible]756	14[illegible]76	134248	1098	30608	326	308
Fault t_m	14.[illegible]79	8[illegible]82	12[illegible]6	10[illegible]0	3[illegible]2	20[illegible]	456
	16.4179	83.582	12[illegible]6	15[illegible]4	3[illegible]6	19[illegible]	396
t_n	3.9801	6.4677	132956	2378	30624	349	299
	0.99503	1.9901	133044	1876	30612	431	364
Fault t_r	17.91104	8[illegible]09	12[illegible]68	15[illegible]6	3[illegible]2	412	2[illegible]1
	18.4408	81.582	12[illegible]84	15[illegible]4	3[illegible]6	477	2[illegible]8

Major Interfaces
Minor Interfaces (Conditional)
False Alarm Interface

Fig. 3. A sampling of metrics and data monitored at a client side

We use these interfaces to catch the abnormality caused by the fault. All these 5 interfaces show significant difference when compared to normal value. The second injected fault is at t_k. In this case, there are three major interfaces and two minor interfaces. Major interfaces include val_CL_MEMIF_ACTIVE, val_CL_MEMIF_IN_DIR, and val_CL_MEMIF_IN_TARGET while minor interfaces include var_CL_CPUIF_KERNAL and var_CL_CPUIF_USER. At time t_l, if we consider only minor interfaces, it will cause false alarm because of the similarity of these states to normal states. The interfaces are used to form two types of significant rules that depend on injected faults. For example, the fault injected at t_m results in increase in read load that is related to IO but the fault at t_r results in the increase of write load. From the above, several types of rules can be generated as following:

- $20 \le \text{var}\_CL\_CPUIF\_KERNAL_t \le 40 \quad T \in (t_j, t_j + k)$
- $5 \le ResponseTime \le 40 \quad T \in (t_j, t_j + k)$

- $127845 \le val\_CL\_MEMIF\_ACTIVE \le 139821$, $5134 \le val\_CL\_MEMIF\_IN\_DIR \le 19321$ and $1395 \le \text{var}\_CL\_IOIF\_READ \ \forall T \in (t_j, t_j + k)$ $0132 \le val\_CL\_MEMIF\_IN\_DIR \le 9846$ and
- $20 \le \text{var}\_CL\_CPUIF\_KERNAL_t \le 40 \ \forall T \in (t_j, t_j + k)$ $\Rightarrow M_i' \le Throughput \le M_a' \quad \forall T \in (t_g, t_g + r)$

The first rule explains that when the number of kernel time spent on client received through cpu interface is larger than and equal to 20 and less than and equal to 40 in time interval *k*, it is abnormal. The second and third rules are about response time, number of active memory and dirty memory and value of I/O load read. In Fourth rule, we didn't include it in our experiments but may classify the interactions more. It means that we can achieve high detection rate by revealing their behaviors and states by virtue of their interactions among interfaces. These kinds of rules are applied in the experiments to detect abnormal flows and increase the performance.

4.2 Abnormality analysis algorithm

Our approach for abnormality analysis to achieve self-healing system is anchored in behavior modeling and analysis of system component impact with rule based. Suppose we have λ system attributes (SA). Then, the flow behavior of SA can be represented as

$$SA_{\lambda}(t, \mathrm{R}) = \{ SA_{\lambda}(t), SA_{\lambda}(t+1), \ldots, SA_{\lambda}(t+\mathrm{R}) \} \tag{17}$$

where R is preliminary block to acquire in-control data to determine the mean $\overline{SA}$ and covariance matrix S. These values are used to verify normal interactions with respect to the λ system attributes. The mean $\overline{SA}$ determines the normal region center and the sample covariance matrix S determines the shape of the normal region. We apply this idea to attribute definition such as AR, CR, and ASR explained in section 3.3 to quantify how close/far the current flow state of a component from the normal state for a given fault scenario which quantifies the current flow state of the system component based on the current values of one or more monitored attributes. The normalized abnormality extent degree (AED) with respect to each attribute is defined as

$$AED_{\lambda}(t, P_j) = \left[\frac{SA_{\lambda}(t, P_j) - AR_{\lambda}(P_j)}{\sigma_{SA_{\lambda}}(P_j)} \right]^2 \tag{18}$$

where AR denotes attribute rates to find out the current flow rate for a given target IP address P_j as observed by a interface monitor and $\sigma_{SA_{\lambda}}$ is the variance under the normal operation condition corresponding to flow. $SA_{\lambda}(t)$ is the current value of system attribute λ.

Figure 4 shows the rule-based analysis for abnormality detection algorithm used by our monitoring and analysis agents that compute and evaluate the attributed definition. During the training stage (line 2), we monitor and collect system attributes where R is the preliminary block to obtain in-control data (line 3). To acquire rule set (line 4), we input the

data set into a rule engine (Cohen, 1995) that produce a rule set. After we get the rule set, we configure the key attributes to gauge attribute definition (line 5 and 6). Once the training stage is completed, process q is applied to real data (line 7). In this algorithm, K (line 9) denotes the number of observations and 15 is used for the number of observations. Attribute definition including abnormality extent degree will be computed for any new observation (line 11). If abnormality extent degree is beyond the normal thresholds and $SA(t)$ violates the rule set, then the system is assumed to be operating in an abnormal state, and then the recovery algorithm is activated to carry out the appropriate control actions such as restarting from the initial point and notifying the information to agents (line 12 and 13). Once accumulating K observations, the thresholds will be revised (line 14 and 15).

```
Process p:
1. Ruleset := 0
2. While (Training) do
3.      Monitor&Collect ( SA1(t), SA2(t), SA3(t), ..., SAR(t) );
4.      RuleSet = Acquire_RuleSet ( SA );
5.      Configure_Important_Attributes ( RuleSet );
6.      AED/AR/CR/ASR ← Gauge_Attribute_Definition;
7. EndWhile
Process q:
8. Repeat Forever
9.      For (t = 1 ; t < K) do
10.             Monitor&Collect  ( SA1(t), SA2(t), SA3(t), ..., SAP(t) );
11.             D = Evaluate_Attribute_Definition( SA(t) );
12.             If ((D > Threshold)&&( SA(t) ∈ RuleSet ))
13.                     Anomaly_Analysis & Detection ( SA(t) );
                apply rule and  attribute definition both
14.             If (t = K)
15.                     Revise_Threshold_Weights()
16.     End For
17. End Repeat
```

Fig. 4. Rule-based analysis for abnormality detection algorithm

5. Problem definition

This section illustrates the fault detection problem, including the data source, abnormal loads, training data and testing data.

5.1 Data source

In our evaluation, we use TPC-W, an industry standard e-commerce application to emulate the complex environment of e-commerce application.

As shown in Figure 5, the system is used to run a typical end user e-commerce activity initiated through a web browser and consisting of several TPC-W transactions. It defines 3 types of traffic mixes such as browsing mix, shopping mix and ordering mix and specifies 14 unique web transactions. In our environments, the database is configured for 288,000

customers and 10,000 items. According to the TPC-W specification, the number of concurrent sessions is executed throughout the experiment to emulate concurrent users. The maximum number of Apache clients and maximum Tomcat threads are set to 90. The workloads are generated by the workload generator that varies the number of concurrent sessions and running time from 90 to 300 seconds. Also, we developed abnormal workload generator which will be defined later. It allows us to make and track the system abnormally behavior.

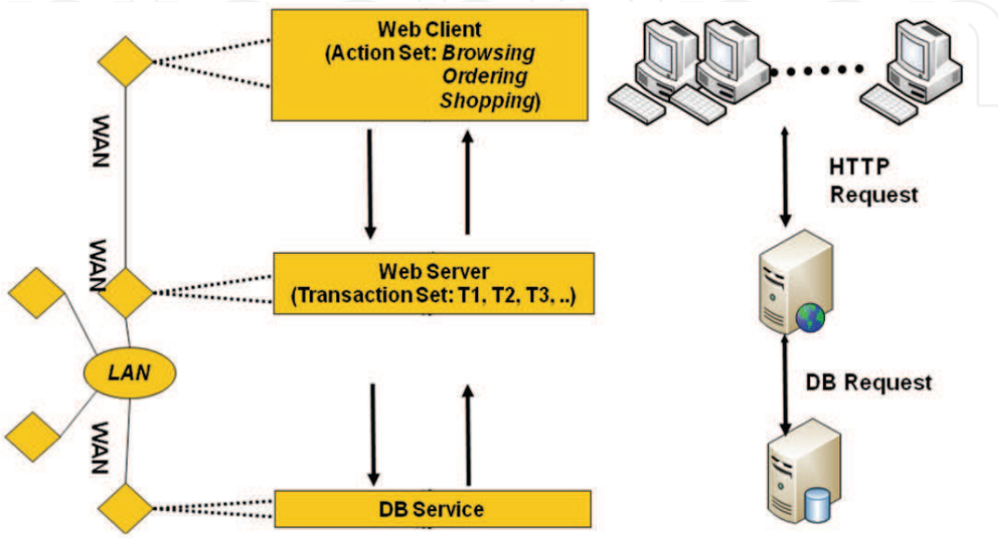


Fig. 5. TPC-W Benchmark E-commerce Application

While every components and workloads are given to the system, we monitor all system interactions and measure different lots of feature including the CPU, IO, memory, operating system, and network devices. The analysis of these features reveals any anomalous behavior that might be triggered by failures in any hardware or software component of the three-tier web based distributed system.

5.2 Abnormal loads

The abnormal loads used in this paper include generally accepted definition (Avizienis et al., 2000), fault and error. If we borrow the concepts, a fault is the cause generating the system corruption and an error that results in the failure is the system state corrupted. We both inject faults such as system corruption and errors such as directly throwing an exception. In our chapter, we usually call fault and error as fault or abnormal loads.

To enlighten our variety in abnormal loads, several papers (Oppenheimer et al., 2003) (Nagaraja et al., 2003) (Chen et al., 2002) are considered as a previous study in faults injected in their experiments. Some of them focus on triggering only application level failures; others inject the faults concentrated on problems that cause program crashes or byzantine faults. We believe that there are system interaction symptoms that characterize how system will respond to a fault injected. Thus, we have confidently decided to include software failures as well as hardware failures in complex distributed systems. Table 2 shows the types of fault classes to be used by our rule based fault detection scenarios. The fault classes can be broadly classified into two groups such as hardware and software. Each group is also divided into three types such as severe, intermittent and lenient.

In these experiments, we inject seven different types of faults explained in Table 2. We categorize and inject these faults by building three different categories. First category is application corruption regarding 3 different types of TPC-W traffic such as browsing, ordering, and shopping. We model faults that are triggered by the interfaces including interactions between an application and the operating system or between an application and other function libraries. These faults injected are from shared libraries into applications to

Fault class	Faults
Software, severe, intermittent	• TPC-W browsing – corruption (Segmentation Fault) • TPC-W ordering – corruption (Segmentation Fault) • TPC-W shopping – corruption (Segmentation Fault)
Hardware, lenient intermittent,	• Network disconnection
Software, severe or lenient, intermittent	• Declared exceptions and undeclared exceptions such as Unknown host Exception • Infinite loops interfering and stopping the application request from completing • DB failure – Access denied

Table 2. Fault cases injected

test the capability of applications to handle faults from library routines referring the fault injection technique [10]. We inject the faults using 3 different system calls such as read (), write (), and close () and observe the effect of injected faults related with interfaces. Hardware faults such as network failure are considered next. It allows us to isolate the node by removing the connection from the network interfaces. Third one is about database related failure such as access denial and application exceptions such as declared exceptions. Because java based e-commerce application engenders various different kinds of failures from programmer faults to IO faults, injection of exception faults are apposite to reveal the abnormal behavior of e-commerce application by tracking system interactions. Here, we injected declared exceptions which are often handled and masked by application itself such as unknown host exception and also infinite loops interfering and stopping the application request from completing. All these faults happen in the process of TPC-W transaction. We believe that the selected faults span the axes from expected to unexpected/undesirable behaviors and divulge the relationship of system interaction for the problems that can occur in a real life.

5.3 Training and testing data

In this study, we have several kinds of data set composed of different number of normal flows and abnormal flows. Our experiments are composed of four kinds of classes such as trustworthiness for fault, noise and data types and performance validation for testing data sets. These training data sets and testing data sets are gathered by tracing normal and abnormal activities. Normal activities and abnormal activities are emulated to produce these data sets by injecting our faults.

First experiment mentioned in section 6.1 is about trustworthiness and validation of our approach for each fault types. We implement and evaluate four scenarios. Fault scenario 1

(FS1) focuses on faults triggered by application corruption using three different types of TPC-W traffic such as browsing, ordering , and shopping. The data set used in scenario 1 consist of 23355 normal flows and 347 abnormal flows. Fault scenario 2 (FS2) considers hardware faults such as network disconnection. The data set consisting of 23355 normal flows and 70 abnormal flows. Fault scenario 3 (FS3) considers application and database faults such as declared exceptions, infinite loops, and database access denial. The data set of FS3 contains 23355 normal flows and 230 abnormal flows. Fault scenario all (FSA) includes all faults explained in section 5.2 and the data set consisting of same number of normal flows previously mentioned and 647 abnormal flows.

The data utilized in section 6.2 employs noise curves such as negative noise (NN) by varying the ratio of abnormal flows in the normal set from 10% to 90% incrementing by 10% each and positive noise (PN) by varying the ratio of nomral flows in the abnormal set from 10% to 90% incrementing by 10% each to evaluate the resilience of detecion algorithm and traced the error rate at each noise ratio points. Each NN and PN data set is consists of 650 abnormal flows and 23354 normal flows, respectively.

The experiment explained in section 6.3 reveals the impact of bulk training data set by composing data based on the specification supplied with the industry standard e-commere environments and includes the four scenarios. Data scenario 1 (DS1) consists of the abnormal set containing normal set containing 23354 flows and 650 flows that are using negative noise curves by varying the ratio of abnormal flows in the normal set from 10% to 90% with 10% increment. Data scenario 2 (DS2) also applies the negative noise curves to explore the correlation in trustworthiness with abnormal flows by building more abnormal flows. The data set is composed of the normal set containing 23354 flows and abnormal set containing 650 flows for data scenario 3 (DS3) and the normal set containing 46000 flows and abnormal set containing 650 flows for data scenario 4 (DS4). Both scenarios employ positive noise curves by varying the ratio of normal flows in the abnormal set to explore the correlation in trustworthiness. We use the testing data set consisting of 20144 flows for normal activities and 420 flows for abnormal activities in the validation of classifiers.

6. Experimental results and evaluation

In this section, we evaluated the detection capabilities of our approach using abnormality extent degree and rule-based fault detection algorithm. The failure data was collected through our distributed test environments shown in Figure 6. We can inject several faults that emulate failures in CPU modules, memory, the disk subsystem, and network modules.

To make the system behaviors as real as possible, we use the following six pairs of workload: TPC-W browsing, ordering, shopping, HTTP gif transfer, MPEG video stream, and HTTPS secure transactions. To generate the fault detection rules, we use a popular data mining tool, Repeated Incremental Pruning to Produce Error Reduction (RIPPER) rule learning technique (Cohen, 1995). The generated rules are based on the insight that abnormality can be captured from system interface flows. The comparisons between our detection approach and the other techniques such as SMO and Naive Bayes were showed and explained in our paper (Kim, 2007). In this approach, we train RIPPER to classify the normal and abnormal flows that occurred during the training period and then apply the generated rules to detect the faults that are injected during each experiment scenario.

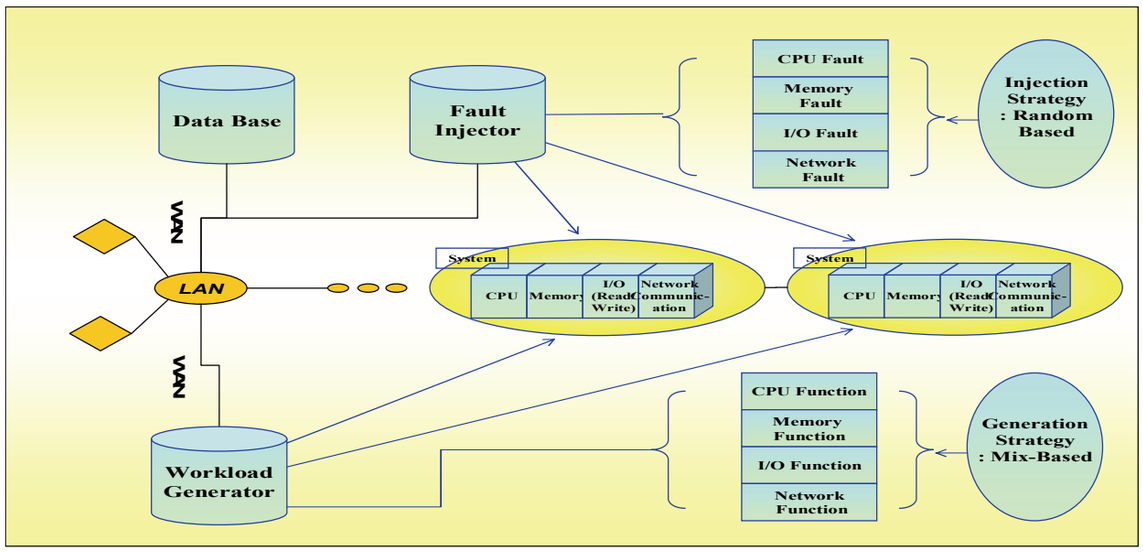


Fig. 6. Testing environments with fault injector and workload generator

6.1 Trustworthiness and validation of rule-based classifiers for fault types

In this experiment, we have three different fault scenarios explained in table 2 and section 5.3. We categorize and inject faults by building three different scenarios. In scenario 1, the faults injected are from shared libraries into applications to test the capability of applications to handle faults from library routines referring the fault injection technique (Avizienis et al., 2000). We injected the faults with three different system calls such as read (), write (), and close () and utilized TPC-W application with three different functions such as browsing, shopping and ordering to observe the effect of injected faults with interfaces. Hardware faults such as network failure are considered in scenario 2. It allows us to isolate the node by removing the connection from the network interfaces. In scenario 3, we chose to inject particular faults such as declared exceptions, infinite loops, and database access denied. For the three scenarios explained, we evaluate the cross-validated true positive rate (the percentage of abnormal flows correctly classified as abnormal) and false positive rate (the percentage of normal flows incorrectly classified as abnormal) to evaluate detection algorithm. To measure the accuracy of generated signatures, we calculated the values for F-measure. We utilized the C implementation of rule algorithm and employed noise curves (negative noise (NN)) by varying the ratio of abnormal flows in the normal set from 10% to 90% at 10% increments to evaluate the resilience of detection algorithm and traced the error rate at each noise ratio point.

Fault scenario 1 (FS1), fault scenario 2 (FS2), fault scenario 3 (FS3), and fault scenario all (FSA) that utilizes all previous three scenarios based on a random mix are given with the percentage of the noise in the abnormal set. Figure 7 shows the false positive rate, true positive rate and F-Measure for all studied scenarios. From the graphs, it is noticeable that FS1, FS2 and FS3 achieved the lowest false positive rate as well as the highest true positive rate. One might think that FSA would have the worst results compare to FS1, FS2, and FS3 since FSA has all of the generated faults that might result in complex and intricate interactions when compared to each fault type. But it is the other way around. Our results

show that FSA has the highest false postitive rate, true positive rate, and F-Measure in many noise values such as false positive rate at 30 % and true positive rate at 20 %. (a) Use Custom Size Format, Width = 17cm, Height = 24cm, (b) top margin is set to 2,5 cm and bottom margin is set to 3,0 cm; left and right margins are set to 2,0 cm on the manuscript format 17x24 cm, (c) use the whole space of all pages, don't leave free space, (d) the text must finish exact at the bottom of the last page. The manuscript has to be submitted in MS Word (*.doc) and PDF format. If you use other word editors and can not transfer it in Word and PDF please contact us.

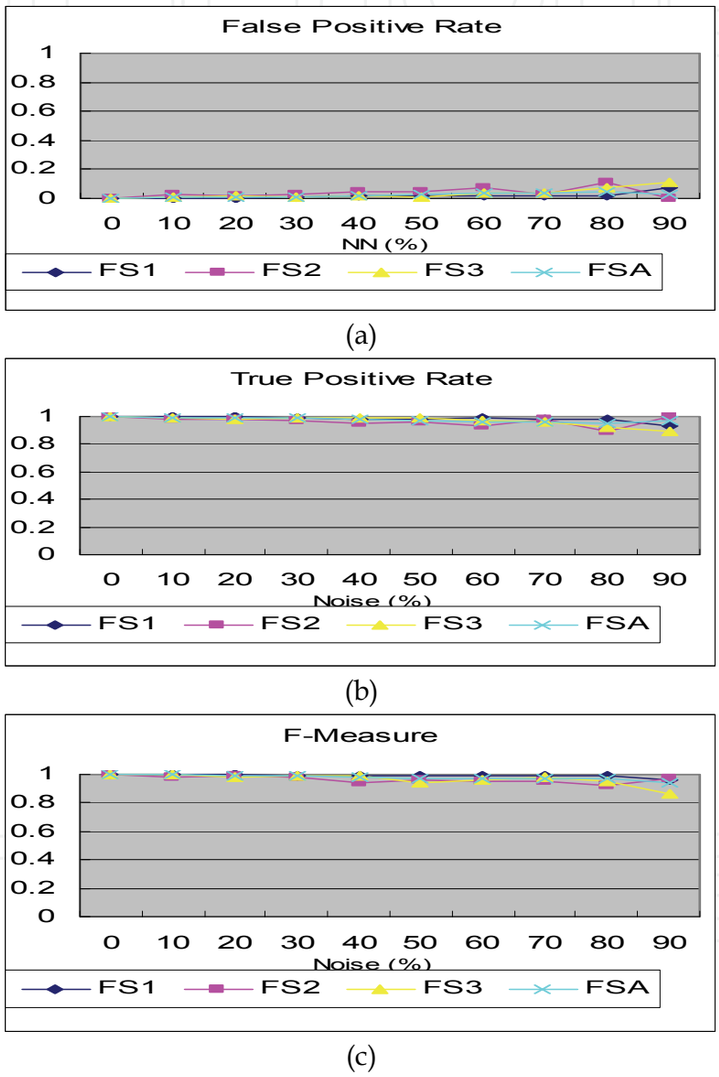


Fig. 7. (a) False Positive and (b) True Positive Rate (c) F-Measure for each fault scenario

6.2 Trustworthiness and validation of rule-based classifiers for noise types

We evaluated the true positive rate and false positive rate to evaluate the detection algorithm. To measure the accuracy of generated signatures, we calculated the values for precision rate.

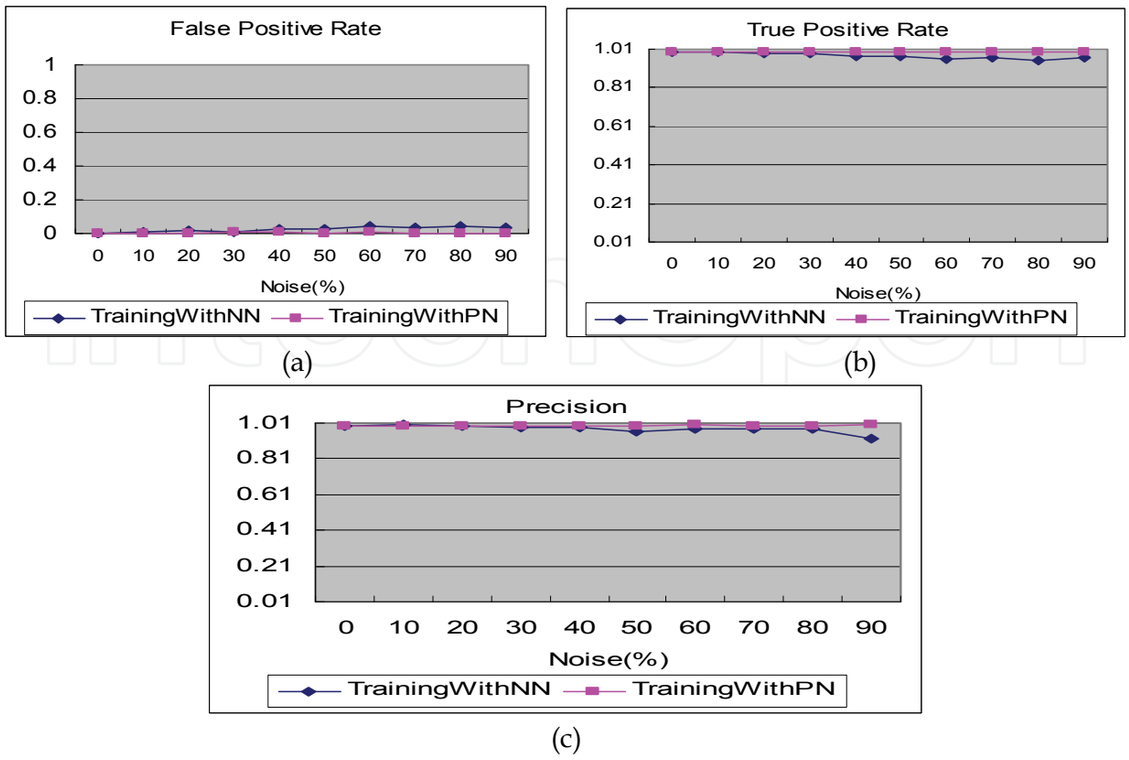


Fig. 8. (a) False Positive and (b) True Positive Rate (c) Precision under varying the noise percentage in the abnormal set and normal set

In this experiment, we employed noise curves (positive noise (PN) and negative noise (NN)) to evaluate the resilience of detection algorithm and traced the error rate at each noise ratio points. The detail description of data set was explained in section 5.3. The ratio of the normal flows in the abnormal set(TrainingWithNN) and abnormal flows in the normal set(TrainingWithPN) are given as the percentage of the noise in the normal set and abnormal set each. Figure 8 shows the true positive rates, false positive rates and precision rate for different noise percentage in the abnormal set and normal set. As shown in Figure 8, our rule based approach for abnormal detection is very reliable even in the nosiy flows. For example, the noise value of 50% produce 0.0265 (NN) and 0.03 (PN) for false positive rate, 0.9735 (NN) and 0.997 (PN) for true positive rate, and 0.959 (NN) and 0.998(PN) for precision rate. These results show that our detection approach is very trustworthy in the distributed computing environment. The performance of the detection algorithm that is trained with TraniningWithPN is more reliable in severe noisy flows. For example, the precision value that is trained with TraniningWithPN equals to 0.998 while the precision value that is trained with TrainingWithNN equals to 0.959 in 50% noise value. The difference between the two noisy environments results from the size of the training set which consists of 23354 normal flows and 650 abnormal flows. However, the 0.959 rate is superior when compared to the other algorithms.

6.3 Trustworthiness and validation of rule-based classifiers for data types

In this experiment, we classify and compose data based on the specification supplied with the multi-tier web benchmark by building four different scenarios. These four scenarios

reveal the impact of bulk training data set. In scenario 1, the size of the training data set is 6,506k that consists of 650 abnormal flows and the 23354 normal flows that use negative noise curves by varying the ratio of abnormal flows in the normal set from 10% to 90% at 10% increments. We explore the correlation in trustworthiness with abnormal flows by building more abnormal flows, 1300 flows, in scenario 2. In scenario 3, the data set is made up of 23354 normal flows and 650 abnormal flows. In scenario 4, we explore the correlation in trustworthiness with normal flows in a data set that consists of 46000 normal flows and 650 abnormal flows.

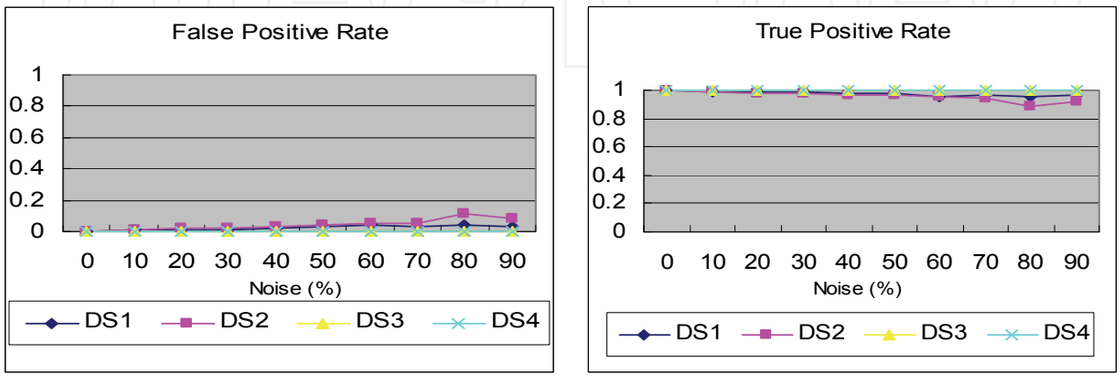


Fig. 9. (a) False Positive and (b) True Positive Rate for each data scenario

Data scenario 1 (DS1), data scenario 2 (DS2), data scenario 3 (DS3), and data scenario 4 (DS4) are given with the size of flows and the percentage of the noise in the abnormal set and normal set. The results are shown in Figure 9. One notable issue is in DS2. All scenarios have good results even in severe noisy flows. However, there is slight difference in DS2 when compared to the other scenarios because of the ratio of the abnormal flows in data set. DS2 has more abnormal noisy flows than the other scenarios. However, we still achieve the lowest false positive rate as well as the highest true positive rate even in DS2. As shown in Figure 9, our rule based approach is a trustworthy even with noisy flows and variance in data sizes.

6.4 Performance validation of rule-based classifiers for the testing data set

Based on the confidence about the results showed before, we tested each result obtained within the generated training data set for variation of the noise percentages, 0% and 10% to underscore the trustworthiness of the detection approach. The testing data set consisting of each abnormal set and normal set was explained in section 5.3. Table 3 shows the false alarm and missed alarm for all the scenarios for different noise percentage in each training data set. As expected, the false alarm and missed alarm rate were very good for fault scenarios such as FS1, FS2, and FS3. But FCA has a higher rate, but is still small. TrainingWithNN and TrainingWithPN achieved very low false alarm and missed alarm even in the noisy situation. Data scenarios also achieved very low false alarm and missed alarm. But the wrong rules generated in the traing stage because of the ratio of abnormal flows in the data result in the increase of missed alarm and false alarm in DS2 with noisy flows. Other than that, the missed alarm and false alarm rate is near 0%. It proves that our approach is superior in each fault scenario as well as the scenario with all faults.

	False Alarm (0% Noise)	Missed Alarm (0% Noise)	False Alarm (10% Noise)	Missed Alarm (10% Noise)
FS1	0 %	0 %	0 %	0 %
FS2	0 %	0 %	0 %	0 %
FS3	0 %	0 %	0 %	0 %
FSA	0.207952 %	0 %	0.399268 %	0 %
Training WithNN	0.207952 %	0 %	0.399268 %	0 %
Training WithPN	0.207952 %	0 %	0.399268 %	0.3 %
DS1	0.208 %	0 %	0.4 %	0 %
DS2	0.208 %	0 %	1.83 %	1.4 %
DS3	0.208 %	0 %	0.16 %	0.3 %
DS4	0.208 %	0 %	0.06 %	1.23 %

Table 3. False Alarm and Missed Alarm rate of all scenarios under applying the rules produced with varying the noise percentage in training data set

7. Conclusion

In this paper, we developed an effective rule-based fault detection algorithm to detect any type of faults for a distributed computing environment. And we evaluate the false alarm rates of our approach for four different fault scenarios and for different data sizes with varying levels of noise. Our analysis show that our approach is superior when compared to other techniques. For example, the precision value that is trained with Tranining with PN equals to 0.998 in 50% noise value and the missed alarm and false alarm rate is near 0%. We are currently extending our approach to not only detect the fault s once they occur, but also perform root-cause analysis and automatic fault recovery.

8. Referring

Avizienis, A.; Laprie, J.C. and Randell, B., Fundamental concepts of dependablility, *in Third Information Survivability Workshop*, Boston, MA, 2000

Chenand, S. & Billings, S.A., Neural networks for nonlinear system modeling and identification, *in "Advances in Intelligent Control"*, pp. 85- 112, Ed C. J. Harris, Taylor & Francis, London, 1994,

Chen, M.Y.; Kiciman, E.; Fratkin, E.; Fox, A. & Brewer, E. Pinpoint: problem determination in large, dynamic Internet services, *International Conference on Dependable Systems and Networks,* June 2002

Cohen, I.; Zhang, S.; Goldszmidt, M.; Symons, J.; Kelly, T. & Fox, A., Capturing, indexing, clustering, and retrieving system history, *ACM SOSP* '2005

Cohen, W. W., Fast effective rule induction, *in the Proceedings of the 12th International Conference on Machine Learning*, pp. 115-123, Morgan Kaufmann, July 9-12, 1995

Kim, B. & Hariri, S., Anomaly-based Fault Detection System in Distributed System, *in proceedings of the Fifth IEEE International Conference on Software Engineering Research, Management and Applications*, August 2007

Lane, T. & Brodley, C., Temporal Sequence Learning and Data Reduction for Anomaly Detection, *ACM Transactions on Information and System Security, vol. 2, 3,* pp. 295-331, 1999

Lazarevic, A. , Kumar, V. & Srivastava, J., *Intrusion Detection: A Survey, Springer US, Volume 5*, 2006

Maxion, R. & Tan, K., Anomaly Detection in Embedded Systems, *IEEE Transactions on Computers, vol. 51*, pp. 108-120, 2002

Nagaraja, K.; Li, X.; Bianchini, R.; Martin, R. P. & Nguyen, T. D., Using fault injection and modeling to evaluate the performability of cluster-based services, *in the proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003

Oh, N.; Shirvani, P. P., & Mccluskey, E. J.. Error detection by duplicated instructions in super-scalar processors. *In IEEE Transactions on Reliability*, pp. 63–75, 2002c

Oppenheimer, D.; Gananpathi, A. & Patterson, D., Why do internet services fail, and what can be done about it?, *in the proceedings of 4th USENIX Symposium on Internet Technologies and Systems*, 2003

Ray, J.; Hoe, J. C. & B. Falsafi. Dual use of superscalar datapath for transient-fault detection and recovery. *In Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pp. 214–224. IEEE Computer Society, 2001.

Reinhardt, S. K. & Mukherjee. S. S. Transient fault detection via simultaneous multithreading. *In Proceedings of the 27th annual international symposium on Computer architecture*, pp. 25–36. ACM Press, 2000.

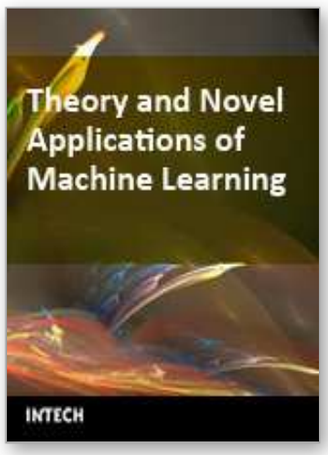
Reis, G.A.; Chang, J.; Vachharajani, N.; Mukherjee, S.S. & Rangan, R.; Design and evaluation of hybrid fault-detection systems, *In the proceedings of the International Symposium on Computer Architecture*, pp. 148 – 159, June 2005

Reis, G. A.; Chang, J.; Vachharajani, N. & Mukherjee, S. S., Software - controlled fault tolerance, *ACM Transactions on Architecture and Code Optimization*, Vol. V, No. N, pp.1–28, December 2005.

TPC-W. http://www.tpc.org/tpcw, April 2005.

Wood, Alan, Data Integrity Concepts, Features, and Technology, *Tandem Division White paper*, Compaq Computer Corporation, 2004

Ye, N. & Chen, Q., An Anomaly Detection Technique Based on Chi-Square Statistic. *Quality and Reliablility Engineering International, vol. 17, 2*, pp. 105-112, 2001



Theory and Novel Applications of Machine Learning

Edited by Meng Joo Er and Yi Zhou

ISBN 978-953-7619-55-4
Hard cover, 376 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009

Even since computers were invented, many researchers have been trying to understand how human beings learn and many interesting paradigms and approaches towards emulating human learning abilities have been proposed. The ability of learning is one of the central features of human intelligence, which makes it an important ingredient in both traditional Artificial Intelligence (AI) and emerging Cognitive Science. Machine Learning (ML) draws upon ideas from a diverse set of disciplines, including AI, Probability and Statistics, Computational Complexity, Information Theory, Psychology and Neurobiology, Control Theory and Philosophy. ML involves broad topics including Fuzzy Logic, Neural Networks (NNs), Evolutionary Algorithms (EAs), Probability and Statistics, Decision Trees, etc. Real-world applications of ML are widespread such as Pattern Recognition, Data Mining, Gaming, Bio-science, Telecommunications, Control and Robotics applications. This books reports the latest developments and futuristic trends in ML.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen