USER VISIBLE FAULT DETECTION FOR E-COMMERCE WEB SITES

A Dissertation

by

PANG-CHUN CHU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2011

Major Subject: Mechanical Engineering

USER VISIBLE FAULT DETECTION FOR E-COMMERCE WEB SITES

A Dissertation

by

PANG-CHUN CHU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Alexander G. Parlos
Committee Members,	Reza Langari
	Won-jong Kim
	Jose Silva-Martinez
Head of Department,	Jerald A. Caton

December 2011

Major Subject: Machenical Engineering

ABSTRACT

User Visible Fault Detection for E-Commerce Sites. (December 2011)

Pang-Chun Chu, B.S., National Chung Hsing University;

M.S., National Chiao Tung University

Chair of Advisory Committee: Dr. Alexander G. Parlos

During the past decade, online shopping has already penetrated into the everyday life as the E-commerce business is growing by leaps and bounds. However, unexpected faults and unplanned downtime of web servers, related to both hardware and software problems continue to cause E-commerce companies to lose revenue. User-visible faults is the major reason E-commerce sites lose revenue and moreover, lose customer loyalty. Detecting user-visible faults sometimes is a difficult task as some of these faults do not produce any error message in the server log files. Therefore, a fault detector which can detect user-visible faults autonomously is needed to support E-commerce site.

End-users appear to change their behavior toward an E-commerce site when they encounter a user-visible fault. This change is reflected in the request transition probability matrix, based on similar mathematical definitions found in state transitions of random processes, such as the Markov processes (chains). This work proposes a system which detects abnormal changes in end-user behavior as reflected in the request transition matrix of web servers. The proposed fault detector is capable of detecting not only changes reflected in the request transition probability matrix but also analyzing the request transition probability matrix to determine whether a real fault or just a special event is taking place. This reduces the false alarm rate. The detector can also localize faults by determining which request and which web page

exhibits the fault.

The detection time for injected faults and the Receiver Operating Characteristics (ROC) curves at a given maximum detection time are used to analyze the performance of the proposed fault detector. The detection results are compared with those obtained by one of the most promising user-visible fault detectors published in the literature. For detection time of more than about 30 minutes, the results demonstrate that the proposed fault detector dramatically reduces the false alarm rate. Furthermore, it distinguishes between real faults and special events. It also provide stable detection times for injected faults irrespective of end-user workload and improves the detection times for two of the four injected faults. At the detection time less than about 30 minutes, the proposed detector's fault detection rate suffers but still with acceptable false alarm rate. At these low detection rates, the comparable detector from the literature provides marginally better fault detection at significantly higher false positive rates. Furthermore, both detectors have unacceptably low detection rates.

A realtime, online version of the fault detection system is also developed in this research. This fault detection system is easy to deploy because it does not require any modifications to the web server source code. Also, it does not produce significant overhead for the monitored web server. Overall, the proposed fault detector has all of the following desirable characteristics: early fault detection, false alarm rate reduction, easy deployment, and effective fault localization.

To My family

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my academic advisor, Dr. Alexander Parlos, for helping and guiding me to finish this research. Without his technical knowledge and patience, this work would not have been possible.

I am very grateful to Dr. Won-jong Kim, Dr. Reza Langari, and Dr. Jose Silva-Martinez. As my dissertation committee members, they really gave me invaluable advice and guidance for this work.

I would also like to extend my appreciation to my friends in Texas A&M University, especially, Aninda Bhattacharya, Parasuram Harihar, Dan Ye, Lin Wang, Yong Li, Tengxi Wang, Gang Li, and Jianxi Fu. They really showed me good time while studying at TAMU. Even through the bad times, they provided me a lot of support and encouragement so that I could continue working on my research and finish my dissertation.

Finally, I would like to express my heartfelt gratitude to my parents, Cheng-ping Chu and Yin-hua Wu, and my sister Janet Chu. They are truly the source of my strength and confidence.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

A. Status of the Current E-commerce Industry

The E-commerce business is growing dramatically in this decade. In 1995, the pioneers of online stores, Amazon.com and ebay.com, were created. Since then, the market for online shopping has been growing with incredible speed. From a report by eMarketer, Inc. [1], in 2000 the revenue generated by the U.S. B2C E-commerce market were $49 billion. In 2005, they reached $148 billion, and in 2010, they exceeded to 225 billion. It is estimated that the growth rate of this market will be about 10% per year for the next three years. This fast-growing trend is not only observed in the United States but also all around the world. According to Forrester [2], in the Western Europe online sales grew 18% from 2009 to 2010, and they will reach $182 billion in 2015. In China, the growth rate is even faster. According to the survey of the IReasearch [3], from 2007 to 2010 the average annual growth rate of the online shopping market in China was 97%, and it will grow by about 30% per year in the next five years.

There are several reasons for the fast growth of the B2C E-commerce market as follows:

1. Convenience and Ubiquity

   Compared to traditional shopping, online shopping is extremely convenient. Almost all of the online stores have high availability (Virtually 24 hours a day, 7 days a week). Also, there is no exact physical location for online shopping. This means that as long as one has Internet access, he/she can shop at anytime

---

The journal model is *IEEE Transactions on Automatic Control.*

and get to any online store all over the world. Via online shopping, people don't have to spend time and money for travelling, worry about finding parking space, or be at stores during business hours. Online stores usually provide powerful search engines. Therefore, customers can find products they are interested in and compare several similar items very easily and quickly.

2. Information Richness

   E-commerce sites usually provide bountiful information for each product they sell and also they have very attractive ways to describe their products. They provide product details not only in text, but also with audio and video. Also the manufacturers' links are provided so that customers can get more information, if they want. Online stores often provide the customer preview section for each product so that shoppers can have ideas about the experience of previous buyers with the product. These two features would be impossible to implement in traditional stores. Some online stores have even online chatting sections so that if customers have questions regarding the products, they can use that section to chat with online salesmen for more information on a real-time basis.

3. High-Quality of Service for Online Stores

   Recently major E-commerce companies such as Yahoo!, Amazon.com, and E-bay.com started offering the hosting services allowing people setup their own online stores. These services usually include user-friendly management interfaces, security protection, and online payment system, allowing people to have their own online stores without much of IT knowledge. It encourages more and more people to sell things online and increase the variety of the online shopping experience. These service providers also organize the information of the sellers so that if someone want to buy a certain item, the web site would list all the

sellers which can provide that item. A shopper can make a decision to choose the seller based on the price, seller's reputation, and the condition of the item. This provides a reliable and diverse environment for online shopping.

4. Improvement of Internet Technology and Transaction Security

   Within this decade, Internet technology has improved dramatically. Fifteen years ago most families still used dial-up modems with maximum transfer rate 56 Kbits/sec to access the Internet but today broadband service with more than 1 Mbits/sec transfer rate is extremely common. This allows E-commerce sites to provide much more abundant and attractive content. Also, with the great improvement of wireless network technology, people can access E-commerce sites anywhere with portable devices such as laptops, cell phones, tablets, etc., in a very secure way. The major concern about online shopping is whether some sensitive personal information such as credit card number would be exposed to the Internet. However, the Secure Sockets Layer (SSL) provides a secure channel for online transactions. Hackers may not get sensitive information easily since during transactions all of the transferred data are encrypted. This increases the credibility of the online shopping experience.

Online shopping has already entered into everybody's daily life. People spend more and more money on E-commerce sites. However, in order to provide better services and handle more customers, web applications and architectures for E-commerce sites have became very complicated. Errors and faults on web servers occur more often than before. This also means that maintaining the high availability of business web site has became more difficult due to the potential faults in the servers. Downtime of web servers is one of the major causes of lost revenue. According to the survey by Infonetics Research [4], the average revenue loss per hour of downtime for

the average businesses in North America is $7,080. The situation gets worse if sites are down during peak times such as holiday session. Amazon.com [5] lost $500,000 revenue in just 20-minute of downtime during Thanksgiving of 2001. Due to downtime, customers can't get into the web site. Also downtime might disrupt and delay orders and supply processing. Moreover, it definitely harms customer loyalty. About 39% of companies [6] claim that unplanned downtime of more than 24 hours would put the survival of their business at risk.

B. Definition of User-visible Faults and Their Impact on E-commerce Sites

Some errors and faults in web sites might propagate and cause site crashes. The downtime of sites would cause companies to loose revenue. Even for faults which don't cause system crashes, they would disable part of web site functionality. This would make end-users feel frustrated because while they send requests to sites, they might encounter unexpected and unsatisfied results. People call these types of faults which result in bad end-user experience while browsing sites, as "user-visible faults".

There are two types of the user-visible faults. The first type is a fault which would be recorded on the error log file or on a the regular log file with error codes. For example, one tries to access a web page which should show the page in detail, but a message "HTTP Error 500 Internal server error" is shown on the page instead. This is because there is something wrong during the web application execution and an exception has occurred. The web server shows this error message to the user because it can't produce an expected result for the request. In this case, the exception that takes place would be recorded on the error log of the web server. Another example would be clicking on a link on of web page. But if the link is a broken link the web server can't find the appropriate page and the message "HTTP 404 not found" is

shown on the page. This fault would be recorded on the request log with the error status code 404. It is clear to see that detecting these types of user-visible faults is not a big problem because such faults would be recorded in the logs of web servers. An administrator reviewing logs could detect and correct such faults.

The second type of user-visible fault is the one which would not produce any program exception or report any error message but end-users still see that there is a problem during online shopping. Fig. 1 shows an example of this type of fault. One checks an item detail page to add this item to a shopping cart by clicking the button "Add to Cart". It is expected that the shopper would be directed to a shopping cart page and this page would show this item inside the shopping cart. However, because of possible problems in the web applications, the item is not actually added to the shopping cart. This type of fault still frustrate end-users because it prevents web applications from fulfilling the requests made by end-users. Usually, it is not easy to detect such faults since they don't produce any error records in the web server log files.

The causes of user-visible faults in web application are as follows [8]:

1. Software failures

   There are logic errors (bugs) in web applications and as an result these web applications don't produce the results properly. For example, there might be a problem inside the code so that it can't compute the shipping fee correctly. End-users might see very weird prices for shipping costs while they try to check out items. Another example is that one tries to check the review section for a item but the page shows nothing. That is because the code for review section refers to non-existent or a wrong table in the database. Some of the software problems would lead to catastrophic failures. For example, if a bug is present

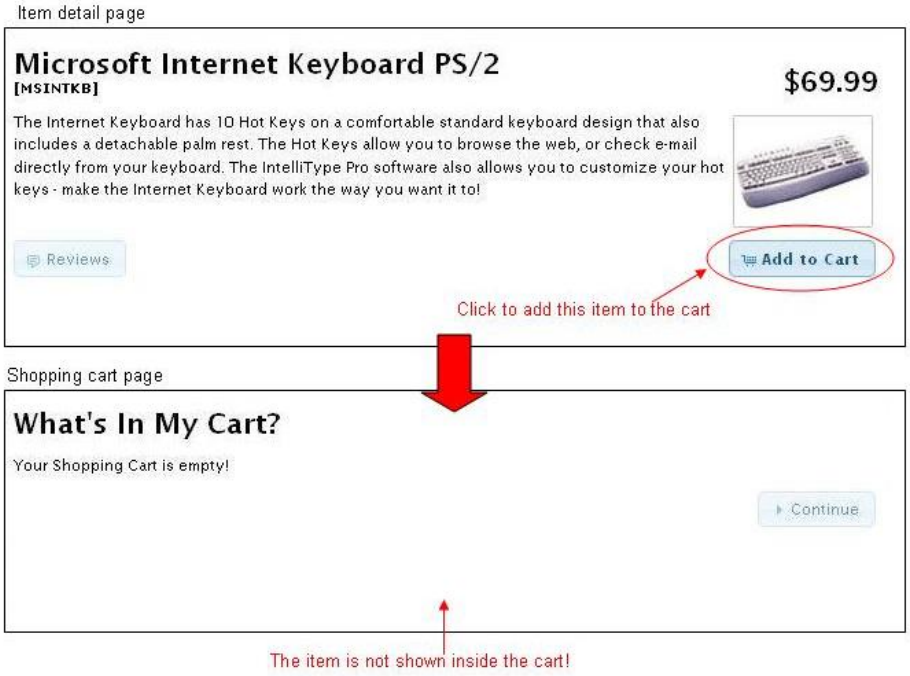

Fig. 1. An example of the user-visible fault [7].

in the program and it tries to access unallocated memory or it doesn't release memory properly, it would lead to exhausting system memory and cause server crash.

2. Database failures

   All the E-commerce sites rely on databases to store data. Some database systems can't handle large amount of data very well and this causes the data or table structures in the database to corrupt. End-users might get null results or wrong results while they want to search some items. Or they might not finish the transactions due to incorrect personal information stored in the database.

3. System hardware failures

   The hardware components of the server, such as hard disks failure or memory malfunction. Usually, hardware failures are severe problems and they lead

servers to crash or freeze. Sometimes environmental problems also cause the hardware failures. For example, overheating and power outages etc. Generally, the hardware redundancy can mitigate such failure.

4. Network failures

   In order to handle large number of requests, an E-commerce site is usually a cluster system with several servers. These servers are connected with network devices. If any of network devices fails, it might make some servers not accessible and overload other servers. End-users might experience slow response from the sites or can't access certain pages.

5. Human errors

   Sometimes operators or administrators of sites make mistakes and cause user-visible faults. For example, they reset the system to its default configuration accidentally. This might cause the end-users not to access some pages because the path configuration has been reset. People might try to check the detail of Item A by clicking the link, but get the detail of Item B because the operator input the incorrect information to the database.

User-visible faults have serious impact on the E-commerce industry. A recent survey [9] reports that during the 2010 Thanksgiving holidays, 24% of customers encountered problems while shopping online and it also estimated that in the same year online retailers lost more than $44 billion due to problems on their sites. Another survey [10] reveals that 33% of customers for online stores found problems during the 2009 holiday shopping season and 41% of the customers who have one or two bad experiences abandoned a retailer's web site and going to competing sites instead.

C. Research Objectives

The E-commerce industry is thriving but faults in web applications is a major issue which causes lost revenue. Such faults might cause system crashes and threaten company survival if downtime lasts long. Some companies use redundant systems so that even if a server is down, a redundant server would be up to eliminate down time. However, the redundancy would double hardware cost of sites and increase overhead of servers due to frequent data transfer between a server and a backup server. Moreover, not all faults would cause system crashes and faults cannot all be fixed by redundancy. Oppenheimer [11] investigated 40 user-visible faults and only 9 of these 40 faults could be fixed by redundancy. Even for faults which do not cause system crashes, they might bring bad experience to end-users and this means the loss of potential customers. Usually web administrators may be able to handle the tasks of fault detection and fault localization for web applications. These tasks must be done very quickly in order to maintain the high availability of sites. However, as the previous section mentioned, sometimes user-visible faults are not easy to detect and fault localization would be very time-consuming, if done manually.

The objective of this research is to implement a user-visible fault detector that can detect faults automatically. The detector must also satisfy the following four characteristics: (1) early fault detection, (2) false alarm reduction, (3) ease of deployment, and (4) effective fault localization. These four characteristics are now discussed briefly:

1. Early fault detection

   A detector should detect faults at a very early stage so that web administrators can address related issues earlier, resulting in fewer end-users impacted by downtime. This can prevent companies from losing potential customers. Also it

can give web administrators more time to fix problems before faults propagate and cause system crashes, thus improving site availability.

2. False alarm reduction

   If a detector generates a large amount of false alarms, this would cause web administrators in tracking down problems that do not exist. Eventually administrators loose confidence in the detection system and ignore its alarms. In the next chapter, several detectors which can detect web application faults, but also consider some of special events, such as promotion activities etc, as faults will be investigated. A good detector should be able distinguish between real faults and special events, so that it does not issue many unnecessary false warnings and waste administrators' precious time.

3. Easy deployment

   In order to satisfy customers needs, an E-commerce system might become very complex. Some detectors require large amount of modification of the source code in web applications, and this increases the difficulty of deployment. A good detector should be easy to install and deploy. Even if a system is expanded or its web applications and configurations are modified, the detector should require minor or no changes to continue fulfilling its duties.

4. Effective fault localization

   Sometimes fault localization is a time-consuming task. A good fault detector can assist web administrators in finding where specifically the faults are occurring so that they can fix the problems much quicker and prevent faults from propagating and causing further damage.

D. Proposed Solution

In order to implement a user-visible fault detector, it is observed that a user-visible fault would change how end-users behave while browsing a site. For example, normally after an end-user adds an item to his shopping cart, he/she might try to check out and finish the transaction or find out if there is any other item he/she is interested in. However, if he/she encounter a fault and can't add the item to his/her shopping cart, just like Fig. 1 shows, he/she would probably retry to add the item to the cart several times, and leave from the site feeling frustrated. All these behavioral changes, i.e. attempt to access individual web pages, would be recorded in the log files. Our detector use this information in the logs to detect the fault. It would be relatively easy to deploy such a detector for any E-commerce site because one must add only a probe to the logger module of web servers.

Fig. 2 illustrates the basic concept of the proposed user-visible fault detector. Initially, the fault detector is in the model training mode. The detection system computes the "request transition matrix" and builds up the models necessary to define healthy state of the system. Then the fault detector enters the fault detection mode. The fault detector not only detect any changes inside the request transition matrix, but also analyze the request transition matrix to determines whether a real fault has occurred or just some special event has taken place, and determine which requests are associated with fault, if a fault has occurred. This reduces the false alarm rate of the detector and localizes the faults. In this research, different types of user-visible faults and several special events are considered, and the detector is tested under different loading profiles, i.e. regular daily profiles and holiday session profiles. The performance of the proposed detector is investigated using the Receiver operating characteristics (ROC) graphs and it is compared with other fault detectors

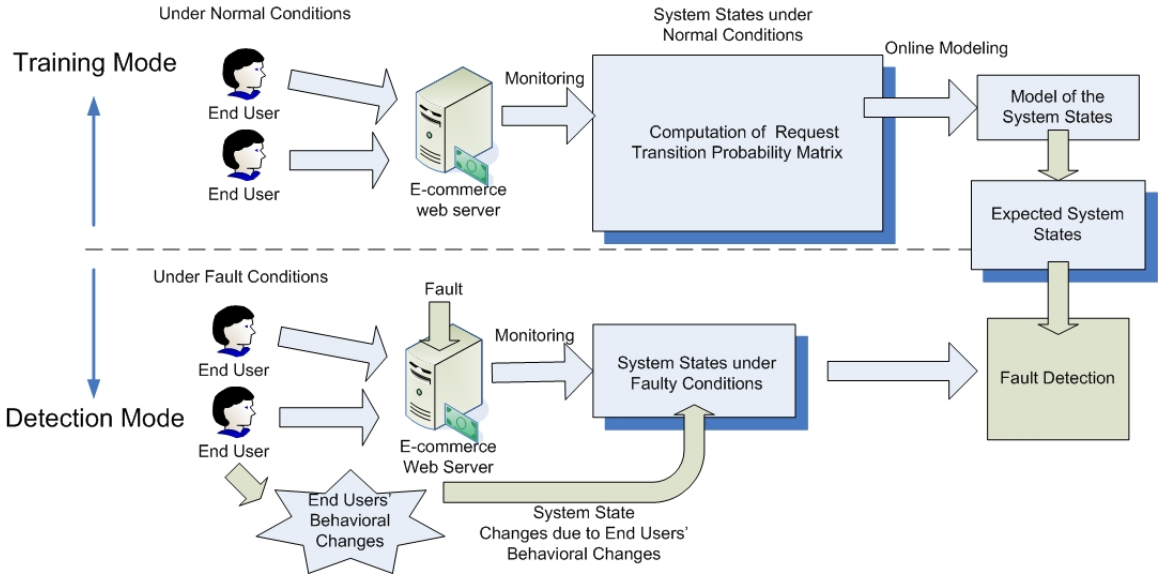

Fig. 2. The proposed user-visible fault detector.

from the literature [12].

E. Research Contributions

The contributions of the current research work are as follows:

1. A user-visible fault detector is developed and tested for E-commerce web sites which has the following characteristics: early fault detection, false alarm rate reduction, easy deployment, and effective fault localization. The ROC curves are used to investigate the performance of the proposed user-visible fault detector and compared to another detector proposed in the lecture.

2. An online fault detection system based on the previous developments is constructed, implemented, and tested, which can detect user-visible faults on a realtime basis.

F. Organization of the Dissertation

This dissertation is divided into six chapters. The current chapter summarizes the status of the E-commercial industry and the problems it encounters. The fault detector using the request transition matrix is proposed in order to mitigate the problems. Chapter II provides a review of existing literature on web server performance improvements, fault detection in networks, and workload development and implementation for E-commerce sites.

Chapter III first explains the basic concepts of fault detection and presents performance metrics and methods of analysis to evaluate effectiveness of a fault detector. Then the motivation to develop the fault detector using the request transition matrix is described. The final two sections of this chapters discusses the architecture and the algorithms of the fault detector using the request transition matrix concept.

Chapter IV mainly discusses the testbed setup for the off-line testing for the fault detection and investigates the experimental results of the off-line testing for two detector (the detector using the request transition matrix and the one using the request file distribution). Four types of faults and three types of special events are considered for the experiments, and the detection time and the ROC graphs of these two detectors are investigated so that their performance can be evaluated.

Chapter V presents the testbed setup for the online detection system and its experimental results. It explains the methods used to deploy the fault detector to a real web server and analyzes the results of the online experiments in order to verify the functionality and the performance of the online detector.

Chapter VI summarizes the research and provides conclusions. This is followed by a brief discussion on the limitations of the research work and suggested future work.

CHAPTER II

LITERATURE REVIEW

Providing good quality of service should be the main goal of administrators of E-commerce sites. This should include both improving performance of sites, such as response time, etc., and detecting faults of web applications more effectively. in this chapter, the research works which build up performance models for web servers and improve performance using control theories are investigated. Then the existing literature related to fault detection in networks is reviewed. First the research works for fault detection with direct approaches are reviewed. For these works, the detectors don't have to build healthy models but try to detect if there are abnormalities within the systems (for example, thread crashes, unconnected node etc.). Then model-based fault detectors are discussed. These detectors have to collect data for training and building the models. After the models are built, the detectors can compare measured data and estimated data from models to determine whether there is a fault. The research works by the Recovery Oriented Computing (ROC) at the UC Berkely and Stanford University group which focuses on developing several novel techniques for building highly-dependable Internet services is also investigated. Finally, several research of workload modelling which can emulate the workloads of real world network for E-commerce systems are discussed.

A. Web Server Performance Modeling and Improvements

Generally in order to building a performance model for a web server, queueing theory is widely used. The M/M/1/K and M/M/C/K queueing systems are two of the most common performance models for web servers [13], [14]. Using these two queueing models, one can achieve expected web performance (response time, throughput, etc.)

by controlling the queue size (K), number of service processes (C), or service rate.

Lu and Abdelzaher [15] developed a first order ARX model to estimate the response time of web servers using a given service rate. In order to get the expected response time, the queueing model predictor is used for estimating the required number of processes reserved for a web server (server process quota), and a P and a PI controller are implemented to adjust server process quota. In [16], the authors extended their research to a web server with multiple different service classes. By controlling the number of processes allocated by each class, a web server can provide expected relative response delay for each class.

Diao, Froechlich, Hellerstein, and Parekh [17] consider the Apache server as a M/M/C/K/N queueing system. The Apache parameter "MaxClients" is adjusted dynamically based on Newton's Method and fuzzy control to optimize the response time. In [18], they investigate how the two Apache parameters "MaxClients" and "KeepAliveTimeOut" impact CPU and memory usage of web servers and design an multiple-input and multiple-output (MIMO) control system for Apache servers using pole the placement technique and the linear-quadratic regulator (LQR) to regulate the CPU and memory usage.

Pradhan, Tewari, and Sahu [19] conducted experiments on a Apache server to identify two of the bottlenecks : CPU and accept queue of servers. They found that if the workload is the persistent HTTP, the CPU is under utilized and accept queue is the bottleneck, But if the workload is SSL processing, the situation is the opposite. Therefore, they designed a controller which includes both accept queue adaption and CPU adaption in order to meet the performance requirement even under different types of workload.

For performance improvement in an overload web server, Schroeder and Harchol-Balter [20] change the scheduling policy on the web server from standard FAIR

(processor-sharing) scheduling to SRPT (Shortest-Remaining-Processing-Time) scheduling. The SRPT scheduler puts requests into different queues based on the size of the request files. The smaller a request file is, the higher priority queue it would be placed in. The web server would process requests in a certain queue only if the higher priority queues are empty. Chen and Mohapatra [21] propose a different scheduling algorithm called dynamic weighted faring sharing (DWFS) scheduling. The scheduler put different requests into different queues based on request types. They also introduce a measure called "server's productivity" which is defined as a function of request completion and request error rate for a server. Based on the measured server's productivity, the system would assign different amount of CPU time to serve requests for different queues to maintain the performance criterion while a server is under an overload situation.

B. Fault Detection in Networked Computer Systems

For fault detectors using the direct approaches, Sultan, Bohra, and Gallard [22] implemented a failure detection and recovery system in a cluster of Internet servers. They define two sections (Progress Box and State Box) inside the memory of every server in a cluster. Progress Box stores progress counters which are able to indicate the system state of the server and State Box stores the states of sessions of end-users. Every server monitors Progress Box of one of remote servers. If the counters inside Progress Box on the remote server are not changed (this means the system hangs), the monitoring system can move the State Box of the server to another one so that the end-user session can be restored.

Klemm and Singh [23] claim that the mechanism of the JAVA error handler can prevent the whole JAVA server from crashing. However, it can't help in restoring the

availability of the server. Therefore, they developed the Java Application Supervisor (JAS), an extension of the JRE. The JAS can detect faults (e.g., thread hang, thread starvation etc.) automatically and recover from faults by restarting the faulty threads, or suspending/resuming them. This reboot mechanism can actually maintain high availability of the server.

Li, Martin, and Nagaraja [24] developed the Mendosus, a SAN (Storage area network) based test-bed to emulate reasonably large-size networks. Mendosus also provides functions to inject both software and hardware faults into components within the emulated network. The main purpose of Mendosus is to check the performance impact in a cluster system while faults occur. In [25], Nagaraja, Li ,and Nguyen used Mendosus as the test-bed while trying to model the performablity of a cluster-based web system, PRESS. The first phase of this research is to determine the throughput impact under different single-fault loads. In the second phase, they use system configuration and MTTF/MTTR of different faults as the parameters to model the performability of the cluster system. This could provide system designers insights of the performability and availability with different design decisions and environmental parameters. In [26], [27], Fu and Wonnacott used Mendosus to determine the robustness of a JAVA program. The JAVA language provides an good mechanism for the exception handling so that it can provides high level of fault-tolerance if program structure is organized properly. With Mendosus, it is clear to see how an single fault impacts the program code. Therefore, it helps programmers to decide the proper location to place try/catch blocks in order to detect faults and recover from the errors. This could improve the robustness and availability of JAVA-based network services.

For model-based fault detectors, Ward, Glynn, and Richardson [28] measured the request rate at a proxy server as the performance metric and found that the time series of the request rate follows a certain pattern. They used this pattern to build

up the model of the request rate for network failure detection. Deviant behavior (the actual request rate is deviant from the expected one) can reflect network failure. They also found that the auto-correlation of the request rate is strong. Therefore, they developed an enhanced algorithm for failure detection which considers not only the current request rate but also the previous request rates. This algorithm can detect failures earlier, but also causes false positives more frequently. They also measured the number of the TCP connections in the Established state, "Syn_Sent state", and "Syn_Recvd state". These three variables help them to localize the causes of network failures.

Trivedi, Vaidyanathan, and Popstojanova [29] developed an SNMP based, distributed monitoring tool. They used this tool to collect operating system resource usages at regular time interval, and also recorded the time of system outages from several UNIX workstations. They found that almost all system outages are related to memory resource exhaustion. Therefore, they focused on analyzing memory related resources. The techniques of single decomposition are applied to filter out the noise and periodic signal and find out the increasing/decreasing trend of these resource usages. The trend for an individual resource usage can help them to estimate the time to exhaustion for that resource and find out which resource usage could impact software aging the most. In this paper, the trend of resource usage is just a function of time. And in [30], they also take the workload into account. They estimate the time to exhaustion more accurately and predict the system outage earlier than the previous method which only considers time as a variable.

Gross, Bhardwaj, and Bickford [31] developed a mechanism for software aging detection for performance critical computers. They measured over 50 performance metrics related computer system load under normal operation as the training data and applied Multivariate State Estimation Technique (MSET) to build up the model

of normal situation for the computer system. The Sequential Probability Ratio Test (SPRT) is used to determine whether the residual error value is uncharacteristic of the model for anomaly detection. Small memory leak was ejected in their experiments to evaluate the performance of detection. The result showed that MSET was able to detect the resource problems very fast and with high sensitivity and reliability.

Cohen, Goldszmidt, Kelly, Symons and Chase [32] measured several system states as metrics for a 3-tier E-commerce web system, filtered out unnecessary metrics, and found the correlation between the rest of metrics and the server-side average response time. A classifier model was built up by Tree-Augmented Bayesian Networks (TAN) to check whether the violation of response time occurs using these measured metrics. The violation means the response time is longer than a specific threshold. This classifier model is useful in diagnosing the performance problem, i.e. which resource causes the performance problem, and for QoS control.

Shereshevsky [33] increased the number of stress programs running on a computer gradually to emulate the situation of software aging and to collect data of the memory related resource usage. They claimed that the degree of factuality of these resource usages would increase due to software aging. They used H*ö*lder Exponent to quantify the degree of the factuality and found that there are several sudden sustained drops in the average value of the H*ö*lder Exponent (they called this phenomenon "fractal breakdown") before the system crash occurs. These fractal breakdowns could help them forecast the system crashes and provide enough time to prevent the actual crash.

Sprenkle, Gibson, Sampath, and, Pollock [34] constructed a system which can record user behavior, the data state for a web application server, and reproduce these behaviors automatically. This system could help web application designers to determine the faults which are hard to detect off-line. They also developed an automated fault detector called Oracle. The Oracle analyzes the HTML output and

determines whether this output matches the expected result. In this paper, five kinds of faults, data storage, logic, form, appearance, and link faults, are injected to evaluate the performance of the oracles.

C. Fault Detection by the Recovery Oriented Computing (ROC) Group

In this section, the research works by the Recovery Oriented Computing group are discussed. These literature are probably the most relevant for the current research. Chen, Kiciman, and Fox [35] developed "Pinpoint" which can detect and localize application-level faults for a large, dynamic Internet service. They instrumented a J2EE server platform to record the component path for each request and also logged status (success/ failure) of the request. Pinpoint compares the component path of the successful request with the one of the failed request and localizes which component causes the fault by data clustering analysis. Since instruments are in the middleware of the J2EE, no application-specific information is required.

In paper [36], they enhanced the fault detection ability of Pinpoint. They assumed that a failed request would follow the abnormal component path, which is a good indicator of the fault. Therefore, they recorded the past normal behaviors of the application server, and built up the model of the component interactions for each component and path shapes for each request in order to detect abnormal behaviors. The results showed that the system can detect major failures, e.g., exception, omission errors, more correctly than the traditional HTTP and HTML monitors. However, it didn't perform significantly better than the traditional monitors for the detection of source code bugs.

In paper [37], Candea employed Pinpoint as a fault detector and developed several strategies of component reboot to build up an autonomous self-recovering appli-

cation server. Ling [38] also developed an application server with the function of the self-recovery and used Pinpoint as a fault detector. However, they employed the Pinpoint to detect low-level hardware failure, e.g., CPU failure, memory failure, instead of application-level failures.

Bodik, Friedman, and Biewald [39] analyzed the log files from an E-commerce website and claimed that the request distribution does not dramatically change during normal condition. A fault could lead an end-user to retry the request because he/she can't get the expected result or prevent an end-user from processing some requests. This would lead the change of the request distribution which could indicate that a fault has occurred. Therefore, a visualization tool was developed to help web administrators to visualize the distribution change through time.

Also a fault detector which can build up the model of the request distribution and detect anomalous behavior for a server automatically was also developed. The χ^2-test and Naive Bayes classification are used to detect if the request file distribution changes. For the χ^2-test, if a web server has n unique types of the request files, the number of hits for each type of request file is recorded for every duration w and the set of the numbers of hits for all types of request files is expressed as a vector $A(t) = [a_1(t)....., a_n(t)]$. The estimated numbers of hits for all types of the request files for a duration w can be obtained from the model of the request file distribution and these estimated numbers can be expressed as a vector $B(t) = [b_1(t)....., b_n(t)]$. Now the χ^2-test can be used to compute the probability that these two vectors ($A(t)$ and $B(t)$) come from different distributions. The test is performed in the following procedures:

1. Compute the *expected value* for each $a_i(t)$ and $b_i(t)$

2. Compute the total χ^2 value of these two vectors:

3. Compute the significance s of the test using the χ^2 distribution with $n-1$ degrees of freedom. An anomaly score is computed as 1-s.

The anomaly score is considered as the fault indicator of this fault detector and based on the anomaly score, the fault detector can determine if a fault alarm is issued.

The fault detector can provide high accuracy for the detection of the major faults and detects faults much earlier than web operators can find them. However, it also causes false positives due to the system configuration changes, e.g., page update, holiday promotion etc.

D. Network Workload Implementation

In order to implement a testbed for fault detection, the characterization of workload of E-commerce sites must be studied. Webstone [40] is the first widely used software to create workload for web servers. It can simulate conditions similar to multiple users accesses the server simultaneously and also allow people to define the probability to access each page on the server. Barford and Mark [41] developed SURGE which implements idle period into user session for the workload. They consider an user session as an ON/OFF process and the duration of the OFF stage of the process represents the idle time. Two kinds of idle time are defined: active idle time, the time between embedded objects of a request, and inactive idle time, the user think time.This ON/OFF process can exactly simulate how a real end-user accesses a web site. The TPC-W [42] is the first benchmark mainly used for E-commerce sites. The first main component of the TPC-W is the System Under Test (SUT) which is a online bookstore site with all of the essential functions of a real E-commerce site, (browsing new items, checking item detail, placing orders..etc.). TPC-W uses a group of Emulated Browsers (EBs) as the workload. Each EB simulates user session and

uses the transition matrix to decide when a sequence of requests should be sent for a session. SPECweb2005 [43] is another web server benchmark, but it is not only for E-commerce sites but also for online banking sites, and vendor support sites.

CHAPTER III

DEVELOPMENT OF THE USER-VISIBLE FAULT DETECTOR

A. Concepts in Fault Detection

Fault detection, by the definition found in [44], is the indication that something is going wrong in the monitored system. Generally speaking, there are two main categories of the fault detection approaches: model-free fault detection and model-based fault detection. For model-free fault detection, some measurements of a monitored system are set as the fault indicators. Thresholds for these indicators are also predetermined. If the value of any indicators exceed the thresholds, it means that a fault situation is detected. A model-free fault detection system is relatively simple to implement. However, since values of fault indicators might vary a lot due to different conditions, environments, or system inputs, thresholds usually are set conservatively, i.e. choose high thresholds to prevent false positives. This might cause the detection system to either miss faults or detect faults too late.

For model-based fault detection system, it is assumed that initially the monitored system is healthy or near healthy and faults might occur sometime in the future. Fig. 3 depicts the structure of a model-based fault detection system. There are two major phases for a model-based fault detection system. In the beginning, the fault detection system enters the learning phase. A proper set of inputs and outputs are selected in order to build up model(s) for healthy baselines. The system would take a certain period of time to collect the data for the model training phase.

After a detection system has built up acceptable one or more models, it enters the fault detection phase. The system uses the measured inputs and the models to obtain the estimated outputs. The estimated outputs and the actual measured

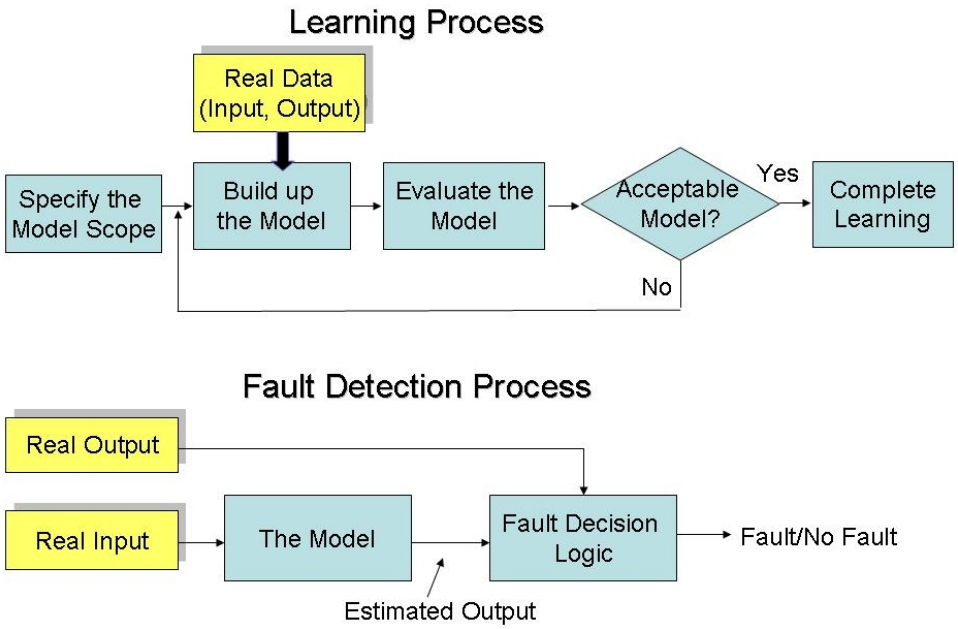

Fig. 3. The architecture of a model-based detection system.

outputs are then compared by the fault decision logic in order to determine whether a fault alert is issued. Compared to model-free fault detection system, the thresholds for model-based system can be set relatively low and the detector can detect faults more effectively.

There are several types of system modeling approaches for use by a model-based fault detection system, as follows:

1. Physical modeling

   If a system's physical behavior is well known, a model of this system can be constructed from basic physical principles, such as Newton's Laws, Ohm's law etc. The parameters of the model have physical meaning and must be properly selected.

2. Empirical modeling

   Sometimes a system is too complex and contains too much uncertainty. It is

difficult to use physical principles to build up a model for it. System identification techniques can be used for this case. Measured data of the system have to be collected for a period of time in order to train a model. The parameters in such models have no direct physical interpretation. Output error model, ARMAX model, and ARX model are wildly used in building up a linear model system. For nonlinear models, neural networks or fuzzy modeling techniques can be used.

3. Distribution modeling

   A networking computer system is highly stochastic and it is not easy to use system identification techniques to build up dynamic models of it. Probability theory and maximum likelihood estimation method can be used to build up distribution models in order to provide good data fit for such system. It is not practical to apply distribution/probability models to system control. However, such models can be used for fault detection.

B. Performance Metrics Used in Fault Detection

1. The Confusion Matrix and the Definitions of Performance Metrics

In order to evaluate the effectiveness of fault detectors, some performance metrics must be discussed. While a fault detector is applied to a monitored system, the confusion matrix [12] can be used to explain the possible outcomes of a fault detector. In Fig. 4, the two columns show the two possible outcomes of the actual system condition, a actual fault occurs and no fault occurs, and the two rows show the two possible outcomes reported by a fault detector, a fault detected and no fault detected. Therefore, for a fault detector operating at a certain threshold, there are four possible outcomes which are shown on the confusion matrix, as follows:

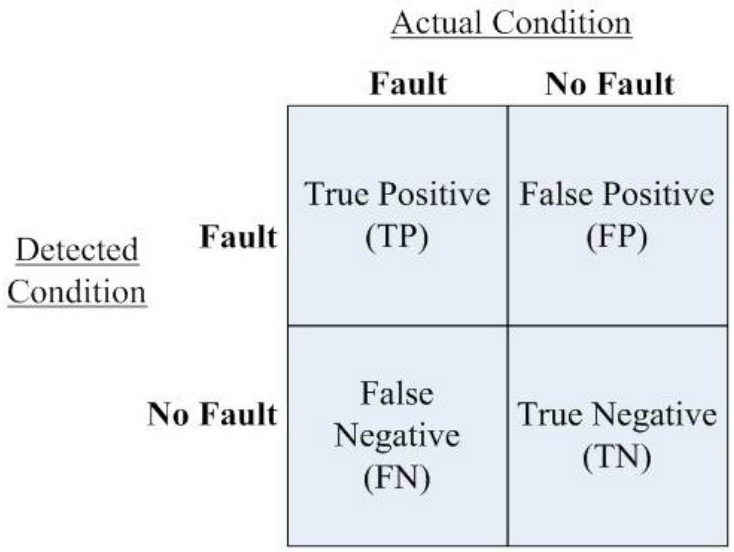

Fig. 4. The confusion matrix of a fault detector.

1. True positive (TP)

   An actual fault has occurred, and the detector detected this fault.

2. False negative (FN)

   An actual fault has occurred but the detector reported no fault (missed fault).

3. False positive (FP)

   A fault has not occurred but the fault detector reported a fault.Also known as false alarm.

4. True negative (TN)

   A fault has not occurred but the fault detector did not reports a fault either.

Only if the detector outcome is a true positive or a true negative, one would consider that a fault detector has made a correct decision. In order to quantify how well a fault detector performs, two important performance matrices are defined as follows:

$$TP\ Rate = \frac{True\ Positives}{Total\ \text{“}Fault\text{”}\ Conditions} = \frac{TP}{TP + FN}\ , \tag{3.1}$$

$$FP\ Rate = \frac{False\ Positives}{Total\ \text{“}No\ Fault\text{”}\ Conditions} = \frac{FP}{FP + TN}\ . \tag{3.2}$$

There are two other performance metrics which are also widely used to evaluate a fault detector, as follows:

$$Precision = \frac{TP}{TP + FP}\ , \tag{3.3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}\ . \tag{3.4}$$

2. The ROC Space and ROC Curves

The Receiver Operating Characteristics (ROC) graph [12], as shown in Fig. 5, is a good tool to help visualize the performance of a fault detector. It is a two dimensional graph with TP rate as its Y-axis and FP rate as its X-axis. A fault detector with a certain TP rate and a FP rate generates one point on the ROC graph. However, the parameters of a fault detector, for example, threshold, can be tuned to obtain different levels of TP rates and FP rates, generating the entire ROC curve. An ROC curve is very useful helping decide how detector parameters should be chosen to optimize its performance.

An ROC graph is also useful in helping compare different fault detectors. The ideal fault detector should have 100% TP rate and 0% FP rate. Therefore, a better fault detector has its ROC curve closer to the top-left corner of the ROC graph. For example, on the Fig. 5 the performance of Detector B is better than that Detector A, because its ROC curve is closeer to the left top corner of the graph. Another good property of the ROC curve is that it is not as sensitive to the ratio of “Faulty Data” to “Not Faulty Data” while it requires both.

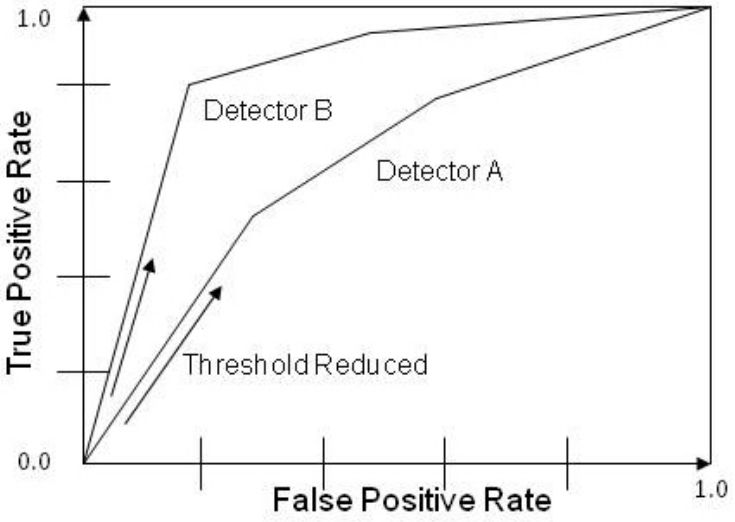

Fig. 5. The ROC graphs of Detector A and Detector B.

C. Motivation for the Request Transition Matrix Concept

1. The Problems of the Current Fault Detectors in Network Systems

The goal of this research work is developing a user-visible fault detector for E-commerce web sites which has early detection, false alarm rate reduction, easy deployment, and fault localization as its four characteristics. As it can be seen from the previous chapter, several fault detectors in networked computer systems have been investigated. However, none of them exhibits all these four characteristics. Sultan's failure detection and recovery system [22] only can check the "liveness" of machines. This means it can't detect a impending user-visible fault before it cause system crashes. Kiciman's "Pinpoint" detection system [36] can detect user-visible faults before the system crashes but it requires insertion of probes to the middleware of web servers. This increase the difficulty of detector deployment. Ward's detector [28] uses the request rate at a proxy server as fault indicator. It is easy to deploy such detector. However, if some special events occurs, for example holiday sales, the

detectors would consider such events as a fault increasing the false alarm rate. Ward's detector also can not tell which request has faults so it can't achieve the goal of fault localization. Bickford's detector [39] only needs to deploy probes to monitor log files and is able to find out the "change of request distribution" for fault detection. However, it has the same problem with Ward's detector: It cannot distinguish between real faults and other special events.

2. The Advantages of the Fault Detector Using the Request Transition Matrix

Now a new user-visible fault detector which measures the request transition matrix is proposed and which includes all four characteristics mentioned. The request transition matrix is defined as a matrix that each of its elements indicates the transition probability from one specific page to another specific page. If an end-user encounters an user-visible fault while browsing a E-commerce site, he/she would retry the requests several time because the server can't fulfill the request. Such behavior change of end-users will be reflected inside the request transition matrix. Therefore, the fault detector tries to detect abnormally increasing number of user-repeated actions and the information it requires are recorded in server log files. It is not difficult to deploy the fault detector since it only requires minor change to the server source code or server configuration. The details about how to deploy such a probe inside log files is discussed in the Chapter V.

The main advantage of the fault detector using the request transition matrix is that it promises to decrease the detector false alarm rate. If end-users find a fault in web applications, this would cause user behavior changes. However, if some special event occurs, for example, items on sales, it would also change the end-users' behavior toward. Both cases would affect both the "request file distribution" and the "request transition matrix". However, only the detector using the request transition

matrix would determine whether the number of users repeating the actions increases. Therefore, the proposed fault detector is able to not only detect such abnormality but also exclude to special events which do not involve increasing the number of request retries. Even for special events which involve increasing number of request retries, the detector can tell if end-users intend to proceed with the repeating actions because of an special event. Therefore, the proposed detector can distinguish between real faults and special events. This means that with respect to false alarm rate reduction, this detector is superior to the detector using the "request file distribution" since the latter detector still considers some special events as faults.

Since the proposed detector can detect a user-visible fault before it propagates to cause a system crash, it has the characteristic of the early fault detection. Also the abnormally increasing number of user repeated actions should reflect in the request transition matrix first and then the request file distribution. Therefore, this detector should detect faults earlier than the one using the request file distribution. Finally, the proposed detector also can achieve the fault localization because it is able to determine which faulty request end-users retry.

D. Development of the Fault Detection Algorithm Using the Request Transition Matrix

In the pervious section, it is mentioned that the fault detector using the request transition matrix actually detects the abnormally increasing number of user repeated actions and also is able to distinguish between real faults and other special events. In this section, the method of model building for this fault detector is discussed. Then several cases of real faults and special events are investigated and the details about the algorithms to detect and identify them are discussed.

1. Model Building for the Fault Detector

Since operation of E-commerce sites is a highly stochastic process, the probabilistic model is used to obtain the healthy baseline of the system. If at any given time there are n types of requested pages on an E-commerce site, then the "request transition matrix" would be an n-by-n matrix as follows:

$$P = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1n} \\ p_{21} & \dots & \dots & p_{2n} \\ \dots & \dots & \dots & \dots \\ p_{n1} & \dots & \dots & p_{nn} \end{bmatrix}, \tag{3.5}$$

where each element of the matrix, p_{ij}, is defined as the probability of transition to page j if the currently browed page is page i.

Therefore, the ith row of the request transition matrix is express as:

$$p_i = \begin{bmatrix} p_{i1} & p_{i2} & \dots & p_{in} \end{bmatrix}. \tag{3.6}$$

This row vector p_i ,the request transition vector, shows the probability of transition to every page from page i, and

$$\sum_{j=1}^{n} p_{ij} = 1. \tag{3.7}$$

Over time the request transition probability matrix P varies, and this is reflected in the algorithm used in computing this matrix. A session is defined as a set of consecutive requests a single end-user sends to a web server. For example, one end-user first goes to the home page of an online store, checks a new item page, sees an item detail, goes back to a new item page, checks another item detail page, and then leaves the server. This session would be represented as:

$\{Home \longrightarrow New\ Item \longrightarrow Item\ Detail \longrightarrow New\ Item \longrightarrow Item\ Detail\}$.

In order to build up the healthy model of the request transition matrix, a certain period time should be defined as the learning time. During this period of time, the fault detection system records all of the end-user sessions of the web server. From the information obtained in these sessions, the fault detector can determine how many times end-users go from a certain page to another page during the predetermined time. Therefore, the probability of transition to every page from a given page can be computed, which allows the development of the healthy model of the request transition matrix.

2. Algorithm for the Fault Detection

After a healthy baseline model of the request transition matrix is built, the fault detector enters the detection phase. A moving window of duration (w minutes) is used for the detector to collect data and compute the request transition matrix. The detector then compare the computed transition matrix with the baseline model in order to detect any possible anomaly. As time progresses the moving window is move by k minutes and the process is repeated.

The fault detector will issue an *abnormal direction* D_{ij} if the following condition is satisfied:

$$\Delta p(t) = p_{ij}(t) - p_{ij}^{m} > T \tag{3.8}$$

where $p_{ij}(t)$ is the probability of transition from page i to page j during the moving window duration w with a leading point corresponding to time t, p_{ij}^{m} is the probability from page i to page j provided by the baseline healthy model, and T is a predetermined detection threshold. An *abnormal direction set* D is defined as the set of all abnormal directions issued at time t.

Several cases about real faults and some special events are investigated, explor-

ing how they impact the request transition matrix. At first, two faulty cases are considered:

Case I: Shopping Cart Fault

For this case, an end-user tries to add an item to his/her shopping cart. However, the shopping cart page shows incorrect information such that the end-user refreshes the shopping cart page several times and then leaves the site because of frustration. Since such fault increases the number of times that end-users refreshes the shopping cart page, the probability of transition from the shopping cart page to the shopping cart page increases. Therefore, the *abnormal direction set* D would be:

$$D = \{D_{ii}\}, \tag{3.9}$$

where i is the ID of the shopping cart page.

Case II: Item Detail Page Fault

If an end-user wants to go to the item detail page from a link of the best seller page for example, and he/she encounters a problem, such as a bad link, wrong items etc., he/she would like to go back to the best seller page and retry the link of the item detail page or try other links to see if this corrects the problem. Such retries will increase the probability from the best seller page to the item detail page and the one from the item detail page to the best seller page. Therefore, if such a fault occurs, the *abnormal direction set* D would be:

$$D = \{D_{ij}, D_{ji}\}, \tag{3.10}$$

where i is the ID of the best seller page and j is the ID of the item detail page.

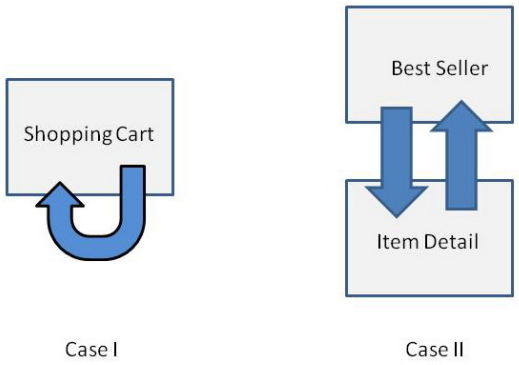

Fig. 6. The abnormal directions of the two faulty cases.

The Fig. 6 depicts the *abnormal directions* of these two faulty cases. It can be seen that for both the two cases, the *abnormal directions* form circles. The fault detector issues an *abnormal circle* C_{ij}, if one of the following conditions is satisfied:

1. Both Dij and Dji exist in the *abnormal direction set* D, or,
2. Dij exists in the D and also $i = j$.

An *abnormal circle set* C is defined as the set of all *abnormal circles* issued during the moving window duration. The main task of the fault detector is to detect whether or not there is any *abnormal circle* occurs. In practice, there could be a large number of such faults that could be encountered in an operational E-commerce system.

The next two cases are special events that might occur during normal server operation:

Case III: New Released Item for Sale

Some important item, for example, a new album from a superstar, the sequel of a famous novel, is just released and this information is shown on the home page of the

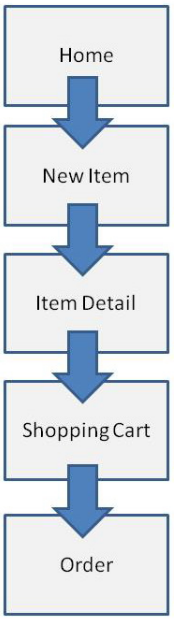

Fig. 7. The abnormal directions of the case III.

online store or announced through the newsletters of the store. This attracts a lot of end-users who want to check this item, put it into the shopping carts, and place the orders for it. The Fig. 7 shows the possible *abnormal directions* if this situation occurs. The *abnormal directions set* of this case would be:

$$D = \{D_{hn}, D_{ni}, D_{is}, D_{so}\}, \tag{3.11}$$

where h is the ID of the home page, n is the ID of the new released item page, i is the ID of the item detail page, s is the ID of the shopping cart page, and o is the ID of the order page.

The fault detector detects several *abnormal directions* while such a case occurs. However, it is clear that these *abnormal directions* don't form one single *abnormal circle*. Therefore, the fault should would not consider this situation as a fault.

Case IV: Clearance Sale

In this case, an advertisement is shown on the home page and it indicates that there is a clearance sale for one specific catalog, for example, Jazz music album. This advertisement attracts end-users to click it and it directs users to the catalog page. The catalog page shows several clearance items. End-users may like to check each one of the clearance items, i.e. go to the item detail page from the catalog page to check the item, go back to the catalog page, and then go to the item detail page for another item, and then place the order to buy some of clearance items.

Fig. 8 shows the possible *abnormal directions* if this situation occurs. The *abnormal direction set* for this case would be:

$$D = \{D_{hc}, D_{ci}, D_{ic}, D_{is}\}, \tag{3.12}$$

where h is the ID of the home page, c is the ID of the catalog page, i is the ID of the item detail page, s is the ID of the shopping cart page, and o is the ID of the order page.

By the definition of an *abnormal circle*, the fault detector detects one C_{ci} because both D_{ci} and D_{ic} are inside the *abnormal direction set*. However, there is one *abnormal directions* D_{hc} toward this *abnormal circle*. This indicates that this *adnormal circle* is formed because end-users "intend" to perform such repeated actions, not because they encounter a fault. Therefore, the fault detector should not consider this case as a fault.

From the previous four cases, it is concluded that the basic algorithm of the fault detector should consist of the following sequence of step:

- Step I: Detect if there is any *abnormal direction* occurs. If it does, go to the step II. If not, no fault alarm is issued.

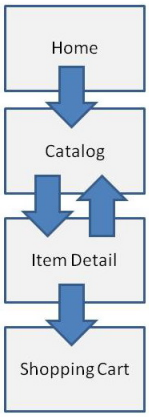

Fig. 8. The abnormal directions of the case IV.

- Step II: Detect if any *abnormal circle* occurs. If yes, go to the step III. If it does, no fault alarm is issued.
- Step III: Find if there is any *abnormal direction* which doesn't form the existing *abnormal circles* and is toward one of the *abnormal circles*. If it does, no fault alarm is issued. If it does not, the fault detector would consider the current situation as a fault and issue a alarm.

Following these steps, the fault detector is not only able to detect faults but also distinguish between real faults and some special normal events. This can prevent the fault detector from issuing false alarms. After a fault is issued, the fault detector would also report the existing *abnormal circles* to the web administrators so that he/she is able to know which pages have the faults. This accomplishes the task of the fault localization.

E. The Architecture of the Proposed Fault Detector

The basic algorithms of the proposed fault detector using the request transition matrix are explained in the previous section. However, there are some problem encountered

if the fault detector uses these basic algorithms. In this section, the problems encountered by the fault detector are investigated. The design of the architecture of a realistic fault detector which overcomes these problems is discussed.

1. Problems of the Fault Detector Using the Request Transition Matrix

The fault detector using request transition matrix detects the presence of any *abnormal direction* and *abnormal circle* that occur within the present detection window. That is basically trying to detect deviations of the transition probability matrix. Since the detector must calculate the transition probability matrix, statistically sufficient number of samples is required in order to obtain the meaningful results. However, it is known that a web server has different hit rates for different requests. Fig. 9 depicts an example of the request distribution of an online store. It is shown that about 14% of total requests are for the home page and only 1.5% are order confirm requests. If the server has total 5,000 requests in a certain period of time, 1,500 of them are the home page requests but only 75 of them are the order confirm requests. This means that for the the order confirm page, there is not enough samples to calculate the transition probability in a statistically significant matter, where for the home page, the number of samples is sufficient.

Given a fixed moving window of duration w, the transition probability matrix is computed at different time as the window moves forward in time. If the number of samples within a window duration is small, then the variation of the transition probability is large even under normal conditions. Fig. 10 shows an example of the deviation of the transition probability from the baseline model, $\Delta p(t) = p(t) - p^m$, where $p(t)$ is the measured transition probability and p^m is the transition probability from the model, for two different pages under normal condition. It is clear that the probability variation of the first plot is much larger than the one of the second plot.

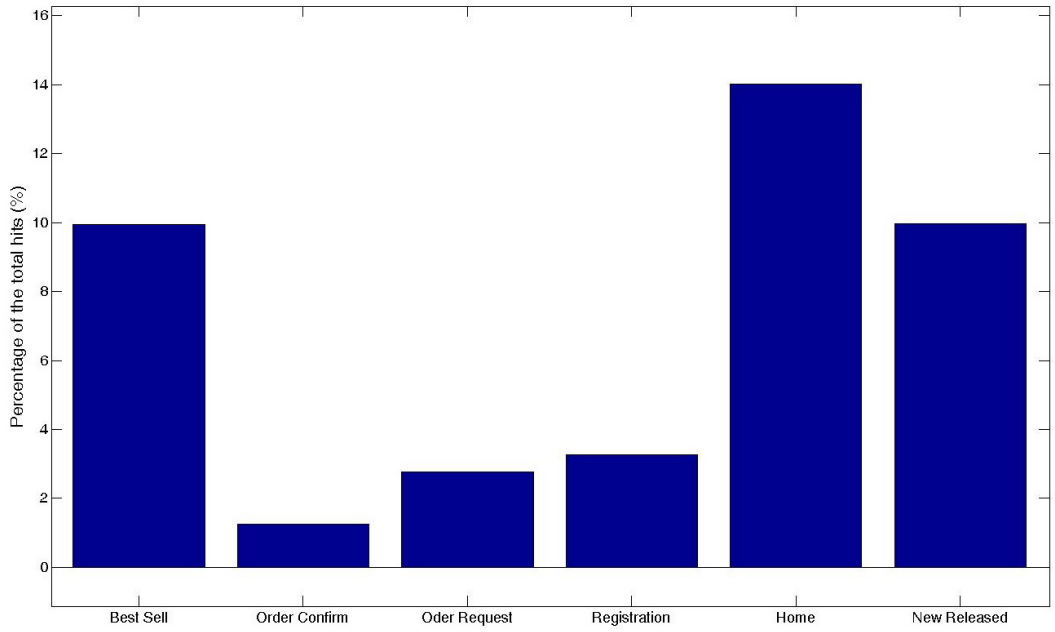

Fig. 9. An example of request distribution of an online store.

That is because the number of hits for the order confirm page is much lower than the number of hits of the home page in a fixed moving window duration. Under such circumstance, it is difficult to set up a single proper threshold for both pages to detect *abnormal directions*. The fault detector would either miss some *abnormal directions* or issue too many *abnormal directions* and cause false alarms. One way to fix this problem is using multiple thresholds. The fault detector could select different threshold based on the number of hits per pages. However, it would be fairly complex to tune such thresholds in order to make the fault detector to meet performance criteria since one has to change many different thresholds all at the same time.

2. Structure of the Fault Detector

Another approach to solve the problem just mentioned is a fault detector using a variable moving window size is proposed. The basic idea is that the fault detector

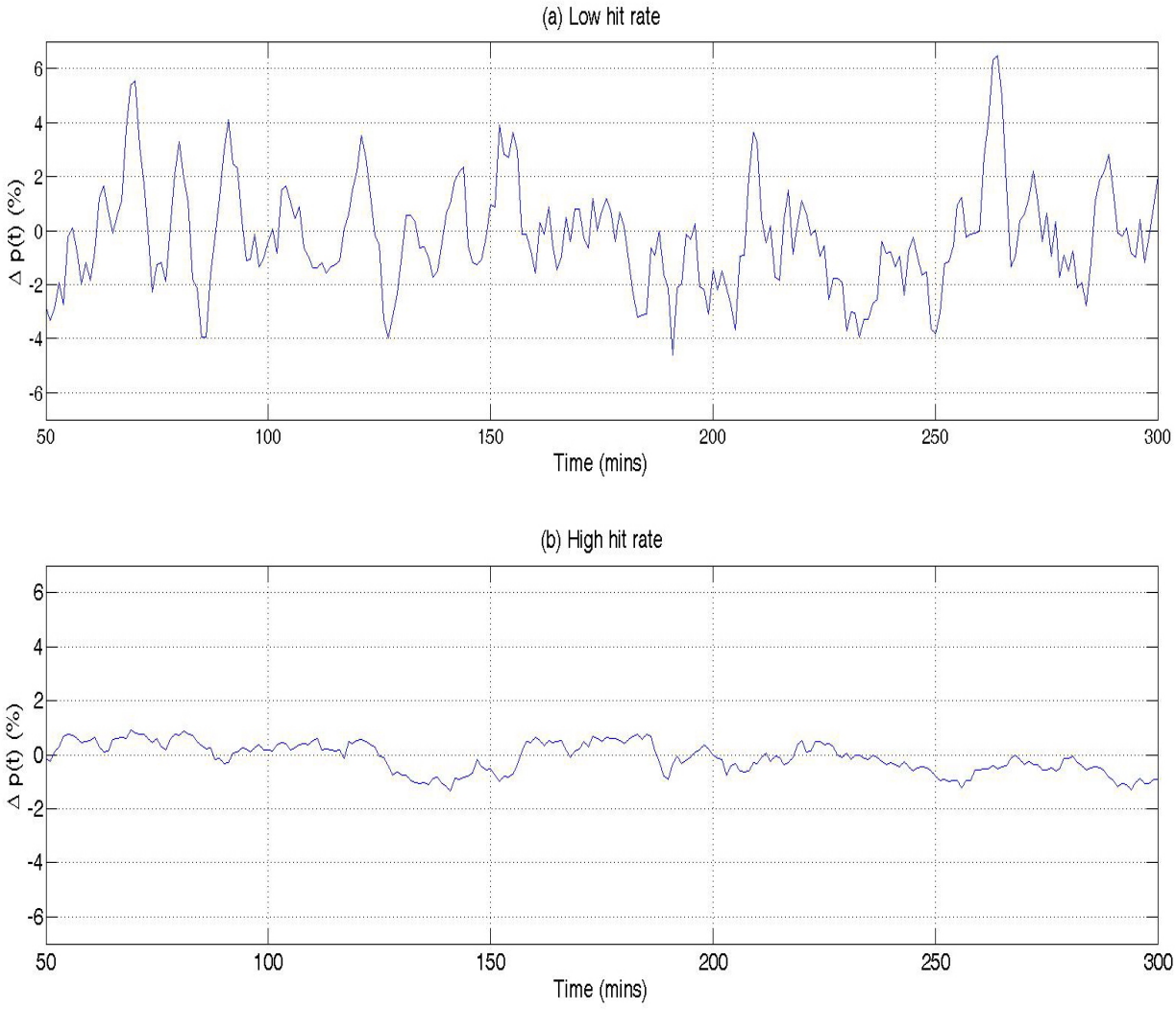

Fig. 10. The deviation of the transition probability ($\triangle p$), (a) from order confirm page to home page, (b) from home page to best seller page.

adjusts the moving window duration (w) for each page based on the hit rate of the particular page. If there are not enough hits for a given page in the current moving window size, the fault detector enlarges the moving window size in order to get enough hits for that page. Therefore, at any one instance, it is acceptable to assign different moving window sizes for different pages.

The fault detector is divided into two different components: the local detection units and the global detection unit. A local detection unit is assigned to each specific web page, whereas the global detection unit is responsible for collecting the information from all the local detection units and handling the fault decision logic. Fig.11 shows the overall structure of the fault detector. In the model training stage, each local detection unit develops a model of the request transition vector (p_i^m) for its own page. Once the model is built, the local detection unit notifies the global detection unit and the global detection unit would put the model into the model database for fault detection. During stage of model utilization, the local detection unit adjusts the moving window size, if necessary, and then compute the current request transition vector, $p_i(t)$ for its own page. The global detection unit provides the corresponding model to the local detection unit so it can determine if there is any *abnormal direction*. If an *abnormal direction* is detected, it notifies the global detection unit and the global detection unit gathers the information from all the local detection units and determine whether a fault alarm is to be issued.

The various components of this fault detector are now discussed in further details:

(a) The Process of Model Building

The Fig. 12 depicts the model building process of the fault detector. The *model threshold* is defined as the minimum number of total hits for a specific page needed to

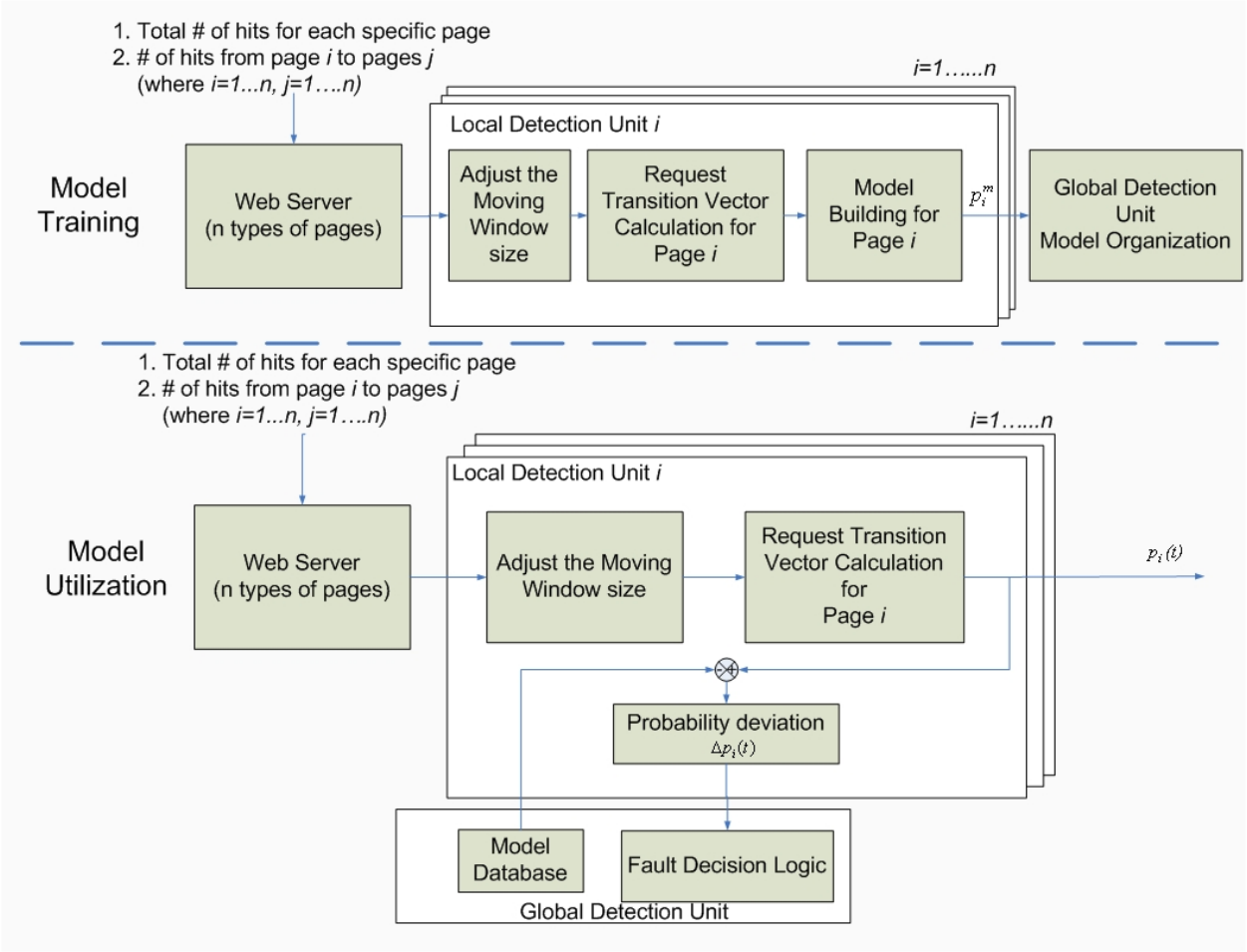

Fig. 11. The structure of the user-visible fault detector.

build up an acceptable model. After the fault detector is started, each local detection units starts collecting data for its specific page. Once *model period* passes, the local detection unit checks if the accumulated number of hits reaches the *model threshold*. If the accumulated number of hits exceeds the *model threshold*, the local detection unit uses the data it has collected to compute the request transition vector (p_i^m) as the healthy baseline model for its own page. Then it notifies the global detection unit and enter the fault detection stage. In the Fig. 12, the hit rate for Page i is larger than the hit rate for the Page j. Therefore, the model for Page i is build earlier than the model for the Page j. The local detection unit for Page i enters the fault detection stage first and it doesn't have to wait for the models of the other pages to be built.

(b) The Processes of Moving Window Adjustment

Fig. 13 shows the process of moving window size adjustment. In order to make the local detection units and the global detection unit work together during the fault detection phase, there are two different timestamps defined: the local detection deadline and the global detection deadline. The local detection deadline is the time when a local detection unit computes the request transition vector for its own page and detects if *abnormal direction* exits, whereas the global detection deadline is the time when the global detection unit gathers all the information from all the local detection units and determines whether or not a fault alarm is issued. After the model for one certain page is built, its local detection unit enters the detection phase. Initially, the local detection unit uses the default window size, i.e. the initial detection period, and the local detection deadline is synchronized with the global detection deadline. The *detection hit threshold* is defined as the minimum number of total hits for a certain

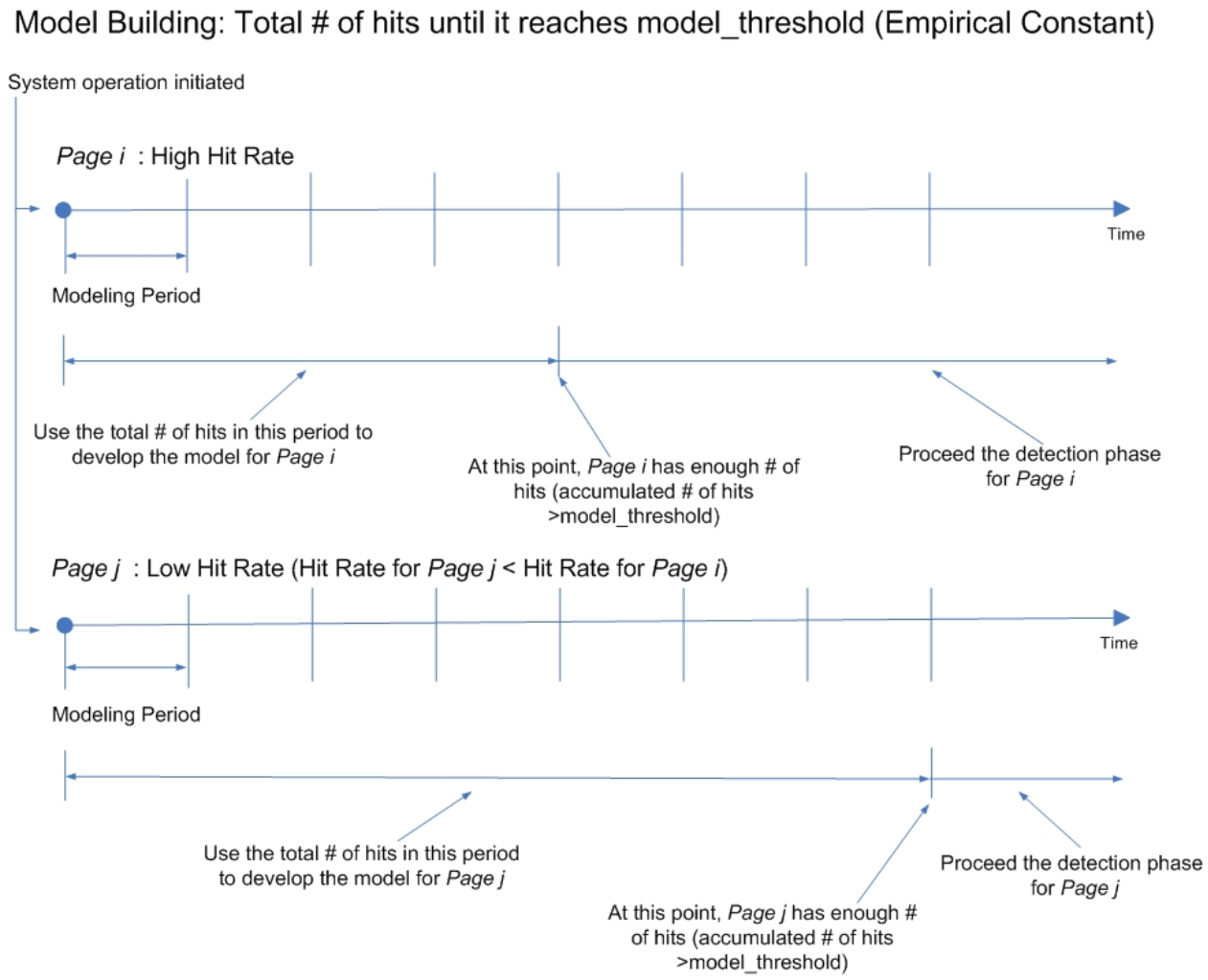

Fig. 12. The process of model building in the user-visible fault detector.

page during a detection period. If the number of hits for a certain page during the detection period is larger than the *detection hit threshold*, the local detection unit does not change the window size. However, if the number of hits is less than the *detection hit threshold*, the window size is enlarged so that number of hits can is maintained at or above the threshold. A local detection unit can also make its window size smaller if the hit rate increases beyond the threshold while the window size is larger than the default window size. No window size which is smaller than the default window size is allowed. A local detection deadline might not coincide with the global detection deadline due to the adjustments of the window size.

Fig. 14, Fig. 15, Fig. 16, and Fig. 17 show two different cases (Fig. 14 and Fig. 15 are for the first case whereas Fig. 16 and Fig. 17 are for the second case) of the window adjustment. In the Fig 14 (a) and the Fig. 16 (a), the hit rate of the former case drops below the *detection hit threshold* whereas the hit rate of the latter case drops but stays above the *detection hit threshold*. Therefore, the window size of the former case is increased as shown in Fig. 14 (b) so that the number of hit during the detection period can stays at the *detection hit threshold*, as shown in Fig. 15 (a). For the latter case, the window size is always the default window size, as shown in the Fig. 16 (b), since the hit rate is above the *detection hit threshold* all the time. Fig. 15 (b) and Fig. 17 (b) depict that the transition probability deviation, are very similar in both cases, since the window size is adjusted for the former case to collect sufficient statistic. A single detection threshold is used for all the pages irrespective of the number of hit rate.

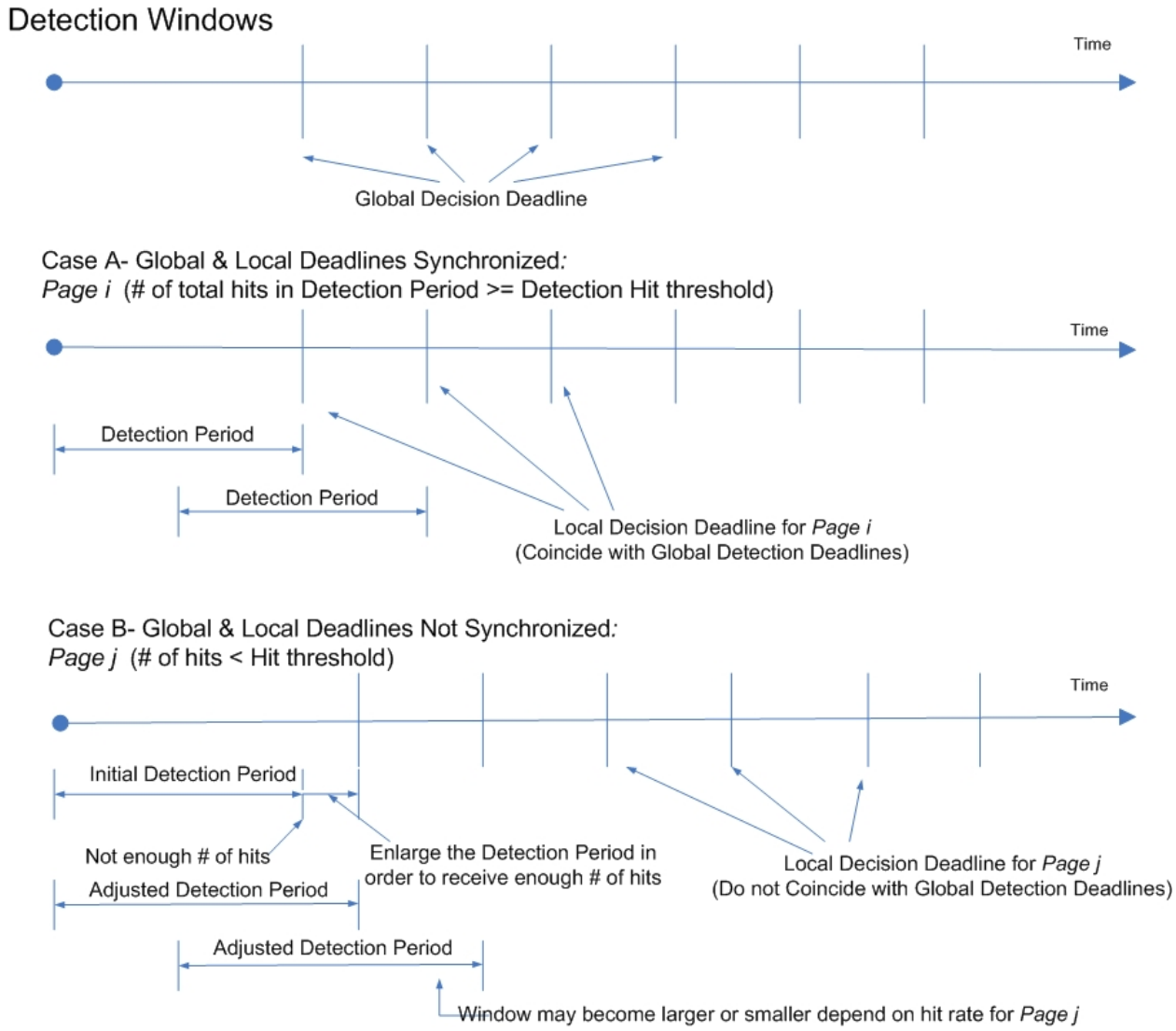

Fig. 13. The process of moving window size adjustment of the fault detector.

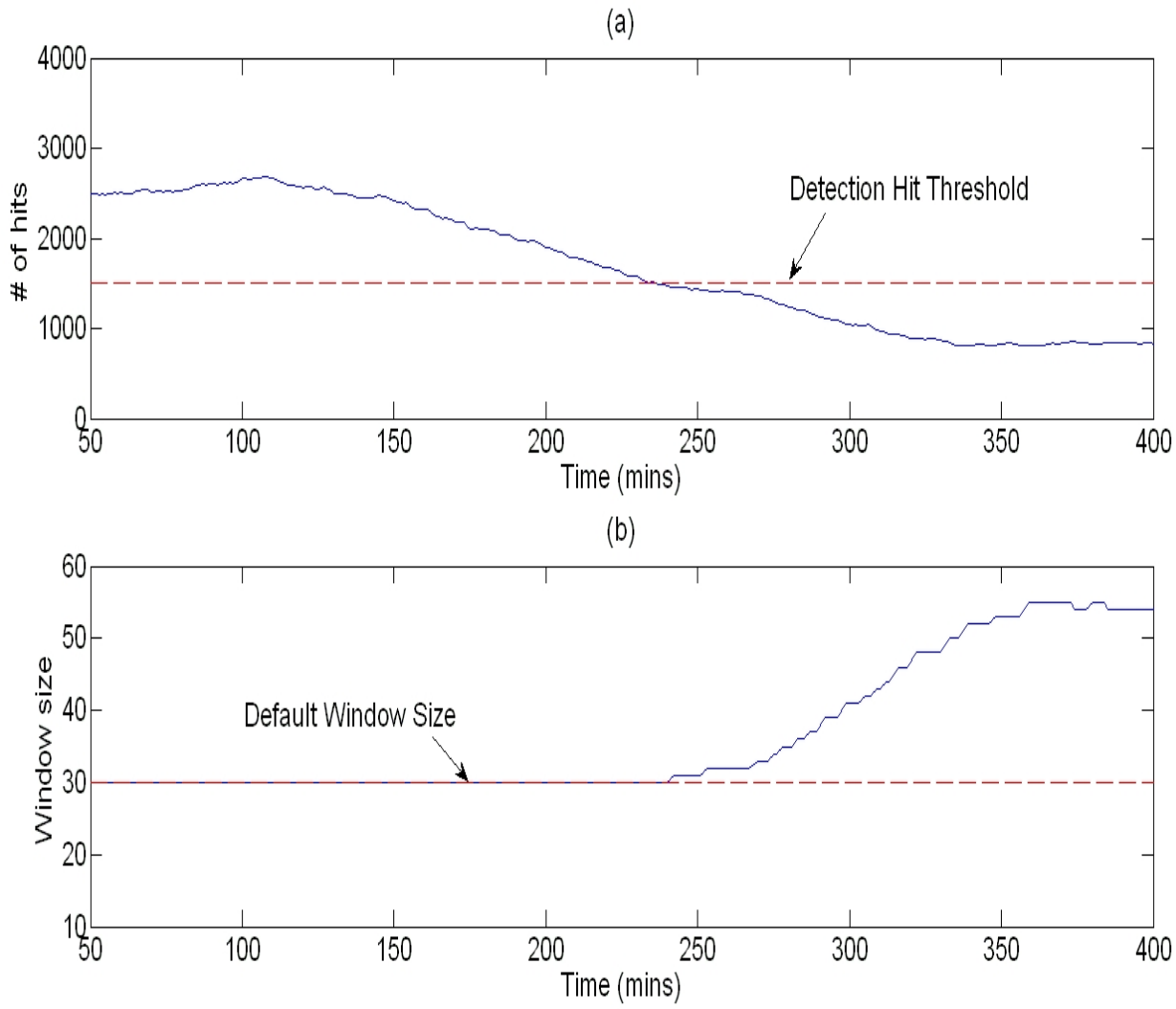

Fig. 14. A case of the moving window adjustment if the hit rate is lower than the detection hit threshold: (a) Number of hits using fixed window size. (b) Window size adjustment.

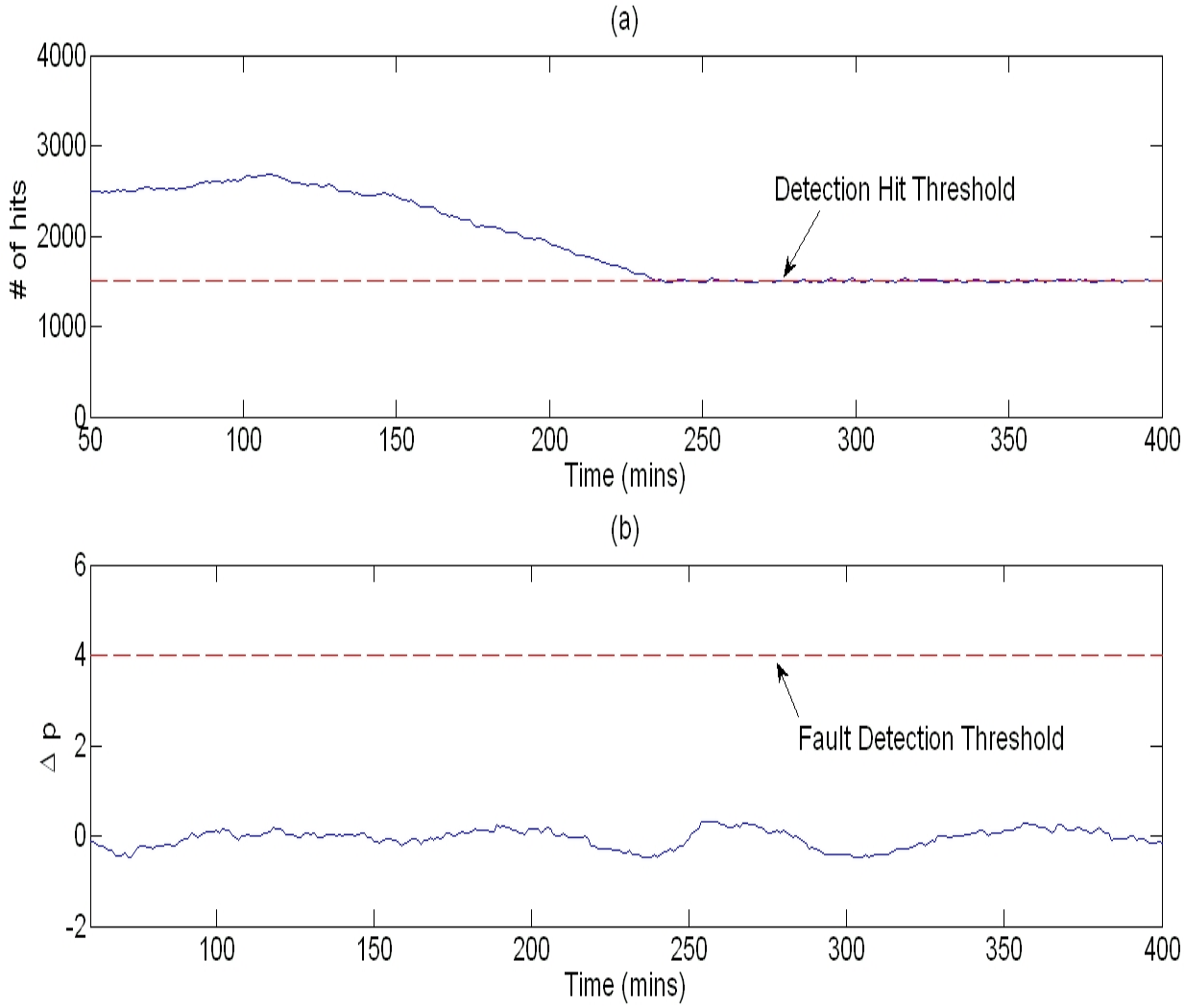

Fig. 15. A case of the moving window adjustment if the hit rate is lower than the detection hit threshold: (a) Number of hits using adjusted window size. (b) Probability deviation ($\Delta p(t)$) from this specific page to another page

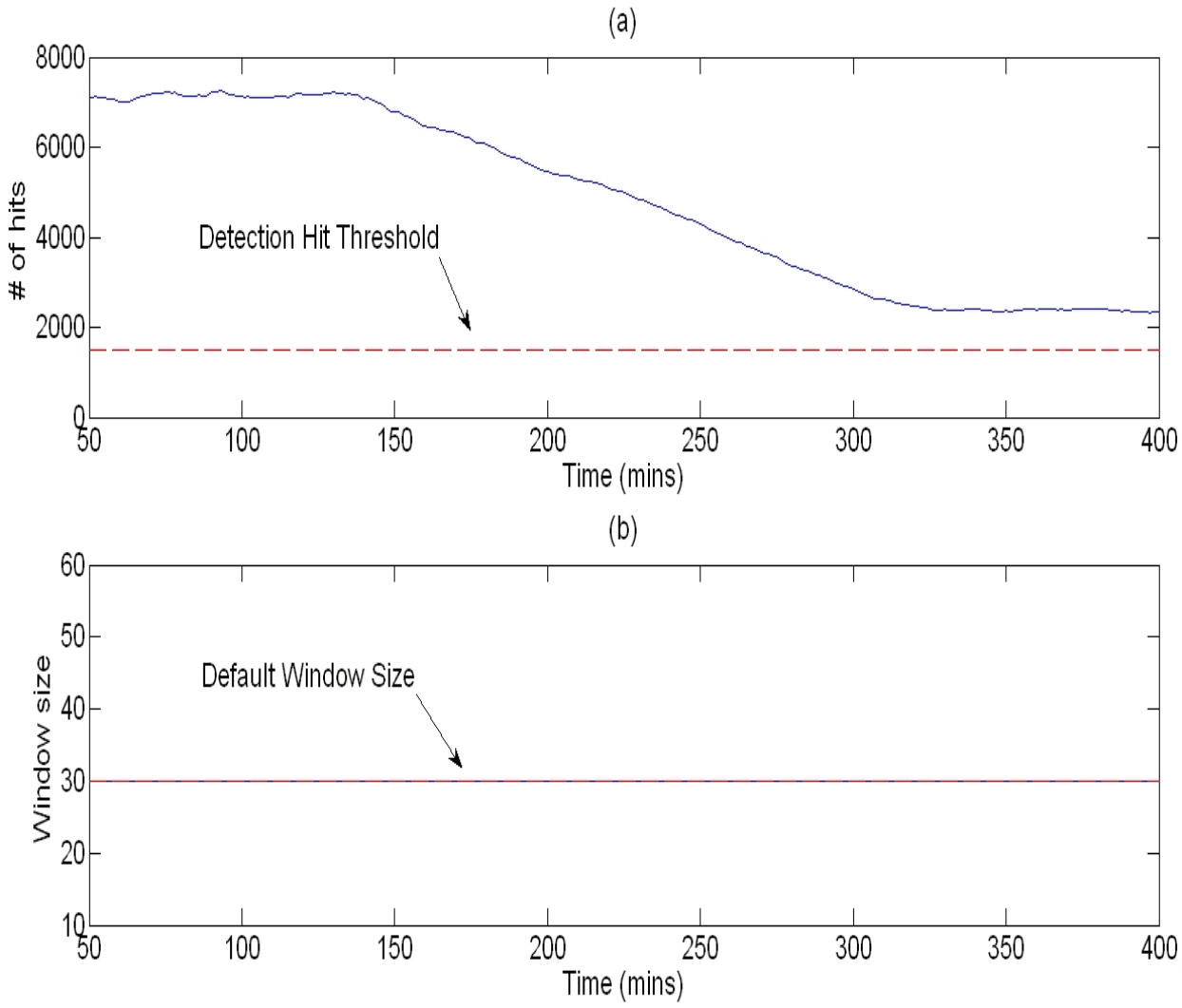

Fig. 16. A case of the moving window adjustment if the hit rate decreases but is always above the detection hit threshold: (a) Number of hits using fixed window size. (b) Window size adjustment.

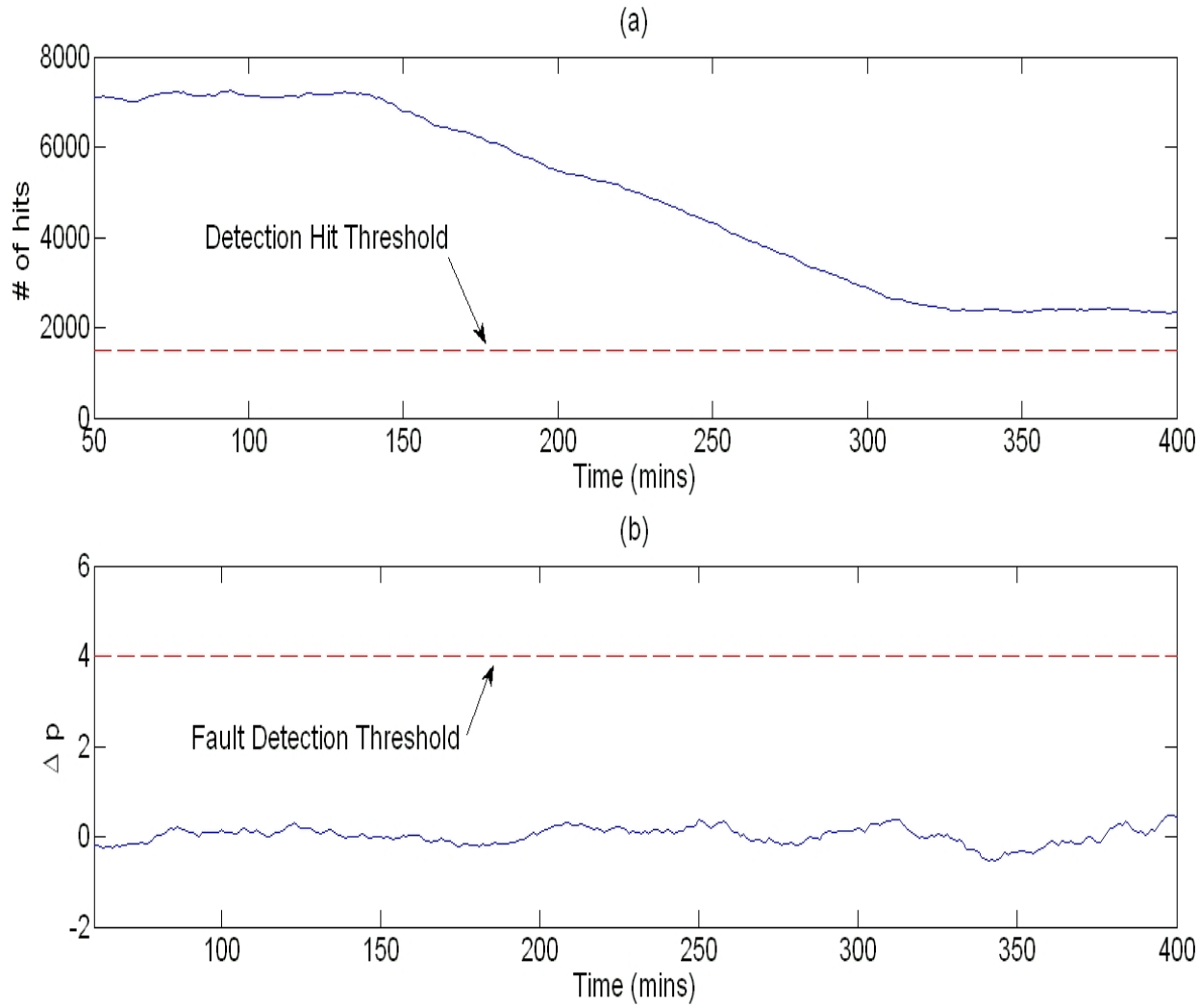

Fig. 17. A case of the moving window adjustment if the hit rate decreases but is always above the detection hit threshold: (a) Number of hits using adjusted window size. (b) Probability deviation ($\Delta p(t)$) from this specific page to another page.

(c) The Processes of Fault Detection

Fig. 18 shows the detection process of the fault detector. It is mentioned that the local detection deadlines for some pages might not coincide with the global detection deadline due to the adjustment of window size. Therefore, the global detection unit gathers all of the information provided by the local detection units between preceding global detection deadline and current global detection deadline to make a decision. Also the global detection unit stores and tracks *abnormal directions* and *abnormal circles* it has detected in order to prevent the detector from making incorrect decision due to lack of information from some pages during the time between the preceding global detection deadline and current detection deadline. For example, in the latest global detection deadline, the detector detects an *abnormal circle* C_{ij} and an *abnormal direction* D_{ki}. The fault detector would not issue a fault alarm because the *abnormal direction* is toward the *abnormal circle*. However, in this global detection deadline, because Page k has a lower hit rate and can't provide information to the global detection unit, the detector only detects the *abnormal circle*. The detector does not issue a fault alarm this time also because it is aware of that the *abnormal direction* D_{ki} is detected in the pervious period and it is not detected now just because of lack of information from Page k.

F. Chapter Summary

This chapter describes the details of algorithm development for the user-visible fault detector using the request transition matrix. In the first section, the basic concepts and the two main approaches of fault detection, the model-free and the model-based, are discussed. In later parts, the confusion matrix is discussed to determine the performance of a fault detector. Two important performance metrics, true positive

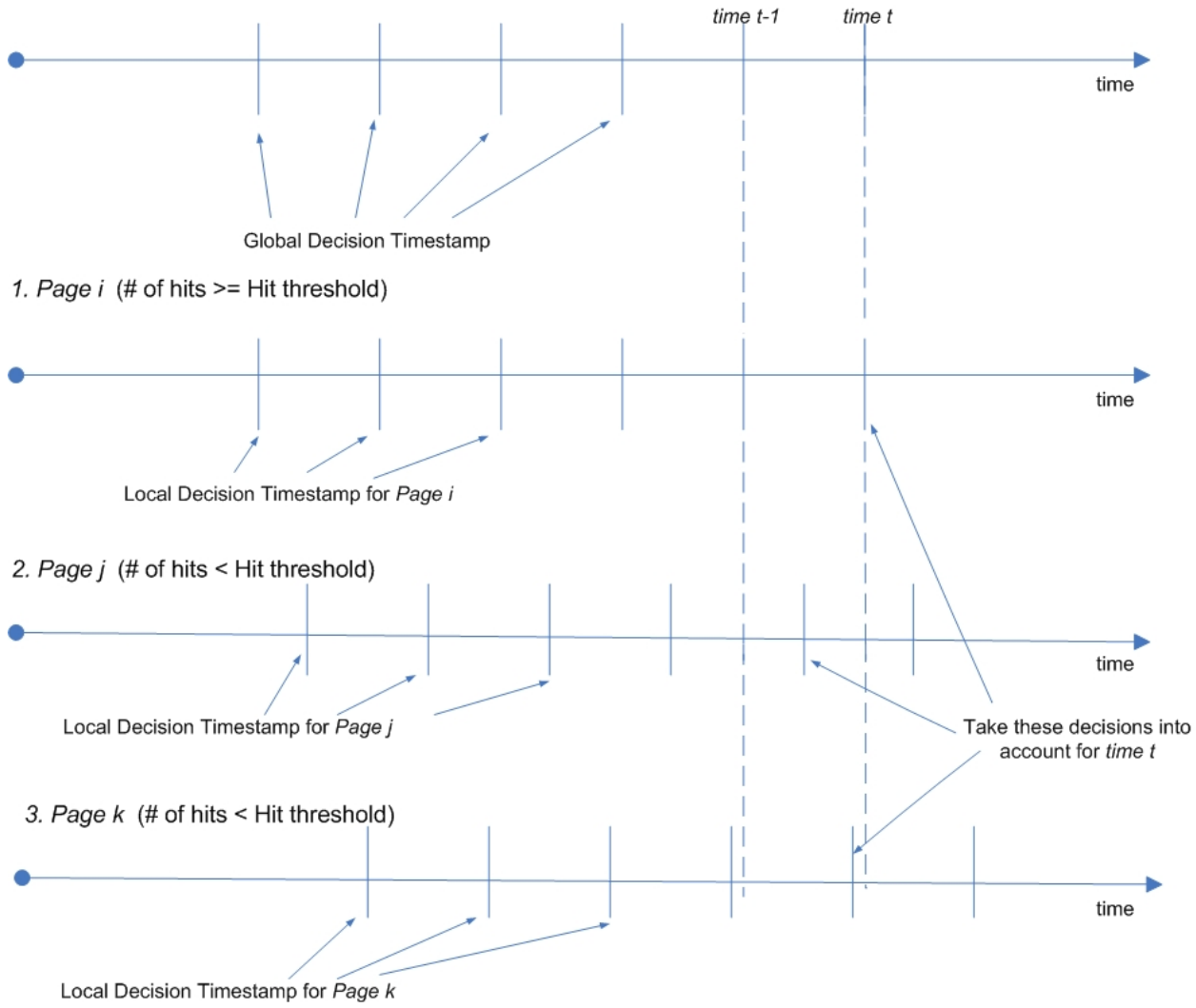

Fig. 18. The detection process of the fault detector.

rate and false positive rate, and the ROC curve are introduced in order to evaluate the performance of the fault detector. These metrics and the ROC curve will be used in the next chapter to compare the performance of different detectors.

The main goal of this research is to develop a user-visible fault detector with four major advantages: easy deployment, false alarm rate reduction, early fault detection, and effective fault localization. The later two parts of this chapter described the details of the fault detector using the request transition matrix and how these four advantages are achieved. The fault detector is easy to deploy since the request transition matrix can be computed from the log files of web servers. It has the feature of early fault detection since it detects end-user behavior change toward web servers and doesn't have to wait until a server crashes. The fault detector can exclude some special events based on *abnormal circles* and *abnormal directions* it detects. This can truly reduce the false alarm rate. Also from the *abnormal circles* it detects, the fault detector can determine which requests have faults so that it can achieve the goal of fault localization.

The final part of this chapter discusses some problems this fault detector faces. Due to different hit rates of different web pages, different fault detection thresholds might be needed for different pages. This increases the difficulty in implementing the fault detector and effectively tuning it. The algorithm with the varied window size is proposed so that the fault detector can use only one single threshold for all web pages.

CHAPTER IV

OFF-LINE TESTING OF THE FAULT DETECTOR

A. Description of the Testbed

A tested E-commerce server and a workload generator must be setup to emulate real-word online shopping and evaluate performance of a fault detector. In this section, an emulated online store and a workload generator which can emulates multiple end-users' behavior is introduced. The software environment and hardware platform of this testbed will be described later in this document.

1. Emulated Online Store and Emulate Browsers

The testbed uses an emulated online bookstore which follows the TPC-W specification [41] as an E-commerce site under test. The TPC-W is a transactional web benchmark which is designed by the Transaction Processing Performance Council [45]. Its online bookstore has all the essential functions an E-commerce site should have. The functions provided by this online bookstore are described as follows:

- Browsing functions: search, product detail, new released page, best seller page, and catalog page.
- Order functions: shopping cart, user registration and login, buy request, buy confirm, and order checking.
- Administrative function: item information update.

The TPC-W also defines eight relative tables inside a database which store all the necessary information for the emulated online store. The emulated online store of the testbed is setup to have 10,000 items and 288,000 customers and all the data are

stored inside the database. The source code of this emulated online store is provided by [46]. Several parts of the source code are modified so that faults and special events can be injected into the online store.

The workload generator of the testbed contains several emulated browsers (EBs) and a remote browser emulator (RBE). Each emulated browser is responsible to emulate online shopping behavior of an end-user and generate a series of requests to the emulated online store. The RBE works as a central controller for all the EBs and it can generate an EB and terminate it. The source code of the workload generator is also provided by [46]. Some modifications are made so that the workload generator can change workload online, in order to emulate situations of real world traffic and an emulated browser can emulate behavior change of an end-user if he/she encounters faults or special events.

2. Description of the Software Environment and Hardware Platform

Fig. 19 depicts the software architecture of the testbed. It is actually a 3-tier system. On the client machine, emulated browsers handle the presentation tier. They work just like real browsers which send HTTP requests to the web server and process responses from the server. On the server machine, Apache HTTP v2.0.63 [47] is used as the web server which handles the HTTP requests from emulated browsers and send back the responses. It only processes requests of static objects, for example, image files, static pages. For the requests which require dynamic contents ,for example, web applications, it forwards them to the application server. Tomcat servlet container v5.0 [48] running on Oracle JDK 1.4.2.15 [49] is the application server of the testbed. It processes requests of web applications from the web server. The web server and the application server together form the logic tier of the system. All the data from the emulated online store are stored inside the database, i.e. data tier. The application

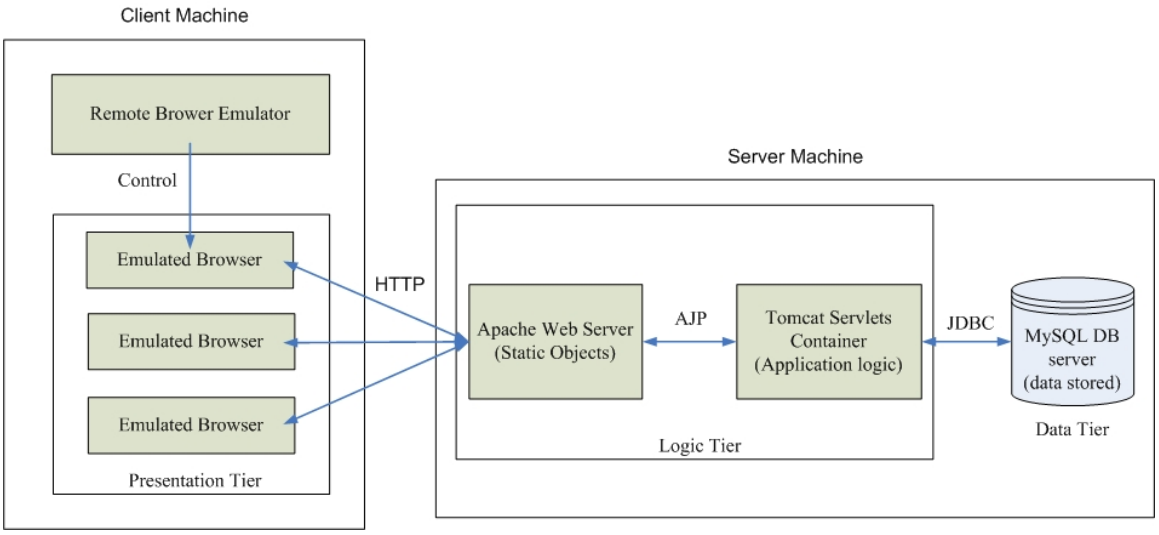

Fig. 19. The architecture of the emulation testbed.

server has to communicate with the database server if it requires data. MySQL community server v5.5.15 [50] is used as the database server of the testbed. The Apache JServ Protocol (AJP) v1.3 [51] is used as the communication protocol between the web server and the application server, and the MySQL Connector/J 5.1.17 [52] is the JDBC driver which provides the interface between the application server and database server. The server machine runs the Linux kernel 2.6.27 while the operating system of the client machine is Microsoft Windows XP professional SP3.

For the hardware configuration, the server machine has an Intel Core 2 Duo E8200 2.67GHz, with 4GB SDRAM, and a West Digital 300GB 10,000 rpm hard disk drive. The client machine has an Intel Core 2 Duo E8400 3.0GHz, with 4GB SDRAM, and a Seagate 500GB 7,200 rpm hard disk drive. They are connected with a switched 1Gbps Ethernet LAN.

B. Description of the Fault Types and the Special Events

Several fault types and special events will be injected into the emulated online store in order to evaluate the performance of the fault detectors.

1. The Injected Fault Types

There are four real fault types used as follows:

(a) Shopping Cart Fault

This type of fault has been mentioned in the previous chapter. An end-user wants to add an item to his/her shopping cart but the shopping cart page shows incorrect information, for example, wrong item in the cart. The end-users might refresh the shopping cart page several times to see if he/she can get the correct result. This type of fault might be caused if the database table which stores the information of shopping carts is corrupted so end-users can not receive the correct information.

(b) Buy Request Fault

An end-user checks his/her shopping cart and press the button "Check out" in order to go to the buy request page. However, he/she gets a complete blank page. He/she might want to go back to the shopping cart page and press "Check out" several times in order to let him/her to place this order. This type of fault might be caused by a bug inside the buy request application. It truncates information of the user sessions so that it can't process user requests.

(c) Search Request Fault

An end-user uses the search engine of the online store to find out some items. However, on the search result page, it shows error message, no result, or a list of unrelated items. He/she might want to retry the search engine several times. This type of fault might be caused by the corruption of the index tables which are responsible for optimizing search performance or a bug inside a web application which mis-allocates memory for the search engine component.

(d) Product Detail Fault

This type of fault was also mentioned in the previous chapter. An end-user wants to go to the product detail page from a link of the best seller page, but gets a problem, for example, wrong item, instead. He/she would like to go back to the best seller page and retry the link of the product detail page or try other links to see if it corrects the problem. This fault might occur because a web administrator enters wrong information of products into a table which stores best sellers and messes up the entire table eventually.

2. The Injected Special Events

Furthermore, three special events which will be injected into the testbed are described.

(a) New Released Item for Sale

An expected item is just released and an advertisement for this item is shown on the home page of the online store. This attracts a lot of end-users who want to check this item, put it into the shopping carts, and place the orders for it.

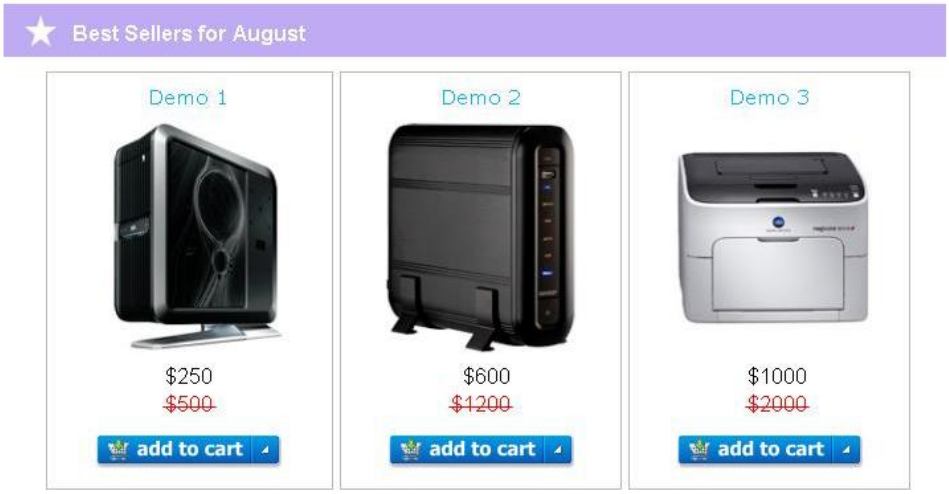

Fig. 20. An example of a page of items on sale [53].

(b) Clearance Sale

An advertisement is shown on the home page and it mentions that there is a clearance sale for one or more catalogs items. This advertisement lures people to click it and it directs people to the catalog page. The catalog page shows several clearance items. Users would like to check each one of the clearance items, and then place an order to buy some of them.

(c) Best Seller Items on Sale

An advertisement is shown on the best seller page and it indicates that there are several best seller items on sale. It attracts people to click it and it directs people to a special product detail page (See Fig. 20). There are several items on sale on this product detail page and this attracts end-users to add these items one by one to their shopping cart and then proceed to check out.

C. Methods Used in Performance Comparison of Detectors

The results presented in this chapter compare the performance of two detectors: the fault detector developed by the ROC group [39] based on detecting the change of the request file distribution using the χ^2-test and the detector proposed in this work based on the request transition probability concept. Several methods are designed and used in this research work in order to compare the two different detectors. The first approach is intended to evaluate the performance of a fault detector on how early this detector can detect a fault. Detection time is defined as the time period the detector takes to detect a fault after it is injected. The first method used in performance evaluation is to compare the detection time of the two detectors for a certain injected fault. The comparison will be under different workloads so that one can observe how detection time of detectors changes as the workload is varied.

The second method is using the ROC curves mentioned in the previous chapter to compare the performance of the two detectors. Two methods are used to generate ROC curves for a fault detector in this research work: one based on detection deadline adjustment and one based on detection threshold adjustment. The detection deadline is defined as the maximin time allowed for a detector to detect a fault. For a faulty case, if a fault detector can not detect a fault before the detection deadline but detects it after this deadline, it is considered as having missed that fault. By tuning the detection deadline, a fault detector can get different pairs of (TP, FP) to form a ROC curve. The second one, the detection adjustment, is widely used to generate a ROC curve for a detector. Buy adjusting the detection threshold, a ROC curve can also be generated for a fault detector. In both approaches, the ROC curves of the two different detectors are plotted on the same figure in order to compare the detector performance.

D. Experimental Results

In order to evaluate the performance of the fault detector using the request transition matrix, the experimental results of the detector using the request transition matrix are investigated and compared with the fault detector using request file distribution with the χ^2-test [39] which is briefly discussed in Chapter II. Both fault detectors have several desirable advantages, such as ease of deployment and fault localization. Therefore, the comparison is mainly focuses on the detection performance of the two detectors, i.e. the detection time, the detection rate, and the false positive rate.

1. Workload Profiles for the Experiments

In order to generate workloads of real world traffic for the experiments, the RBE of the client machine follows the workload patterns which are shown on Fig. 21 to control the number of EBs for daily traffic. These patterns imitate the workload profiles of a real E-commerce site [54]. Two different workload profiles are used for the experiments, the weekday profile and the holiday profile, and nine points, Point A through Point H, at different workload levels are indicated where faults and special events are injected so that the performance of detectors can be investigated under different traffic levels.

2. The Parameter Setup and Model Building

As mentioned in the previous chapter, several parameters must be determined for the fault detectors. They are the model hit threshold, the detection hit threshold, the default window size, and the fault detection threshold.For statistically significant results, 50,000 points are selected as the model hit threshold as this is sufficient number of points to build up a model of the request transition vector. The default

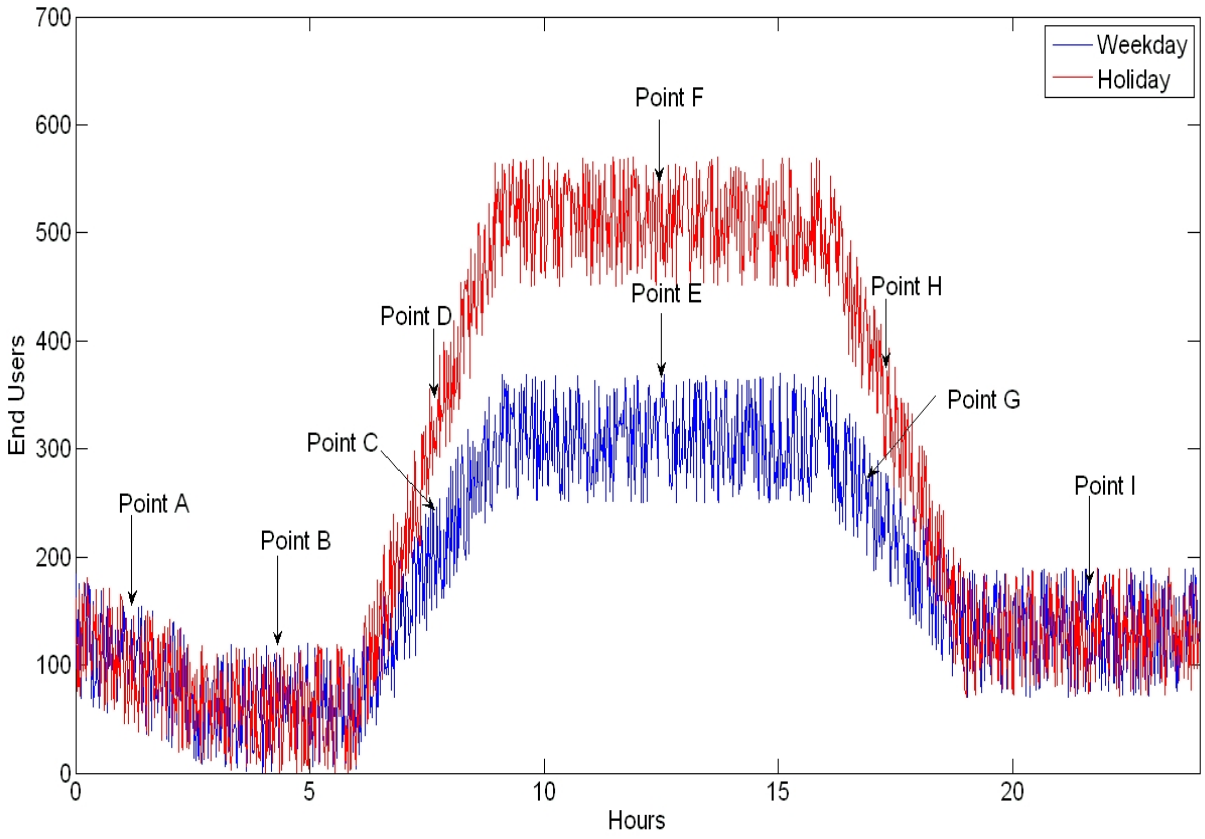

Fig. 21. The workload profiles and the fault/special event injected points.

window size is set at 30 minutes but the window moves only 1 minute at a time in order to obtain better detection resolution.

In order to determine the detection hit threshold and the fault detection threshold, the data from an 36-hour experiment under normal conditions and constant workload are collected. Several moving windows which contain different number of points are selected and the probability deviations for each moving window size are computed from these data. Fig. 22 shows the maximum probability deviations for moving windows which contain different number of points. It is clear to see that the maximum deviations approach to a certain level if the moving window contains more than about 1,500 points. Therefore, 1,500 points is selected as the detection hit threshold as a detection hit threshold larger than this does not help the detector gain more benefit, i.e. lower the fault detection threshold, and only delays the fault detection. The fault detection threshold is chosen because this is just above the maximum probability deviation for a moving window with approximately 1,500 points.

The data of an 48-hour experiment under normal condition and using the weekday workload profile are collected for model building for both the fault detector using the request transition matrix and the one using the request file distribution. The model building for the former fault detector follows the method mentioned in the previous chapter. The model of request transition vector (p_i^m) for a page is built up once the number of accumulated points for the page reaches the model hit threshold. Therefore, the models for some pages don't require the entire 48-hour data to be built up.

In order to determine the fault detection threshold for the fault detector using request file distribution, The 36-hour data used for determining the fault detection threshold of the detector using the request transition matrix are tested using the χ^2 test presented in [39]. The highest score of the fault indicator using the data is around

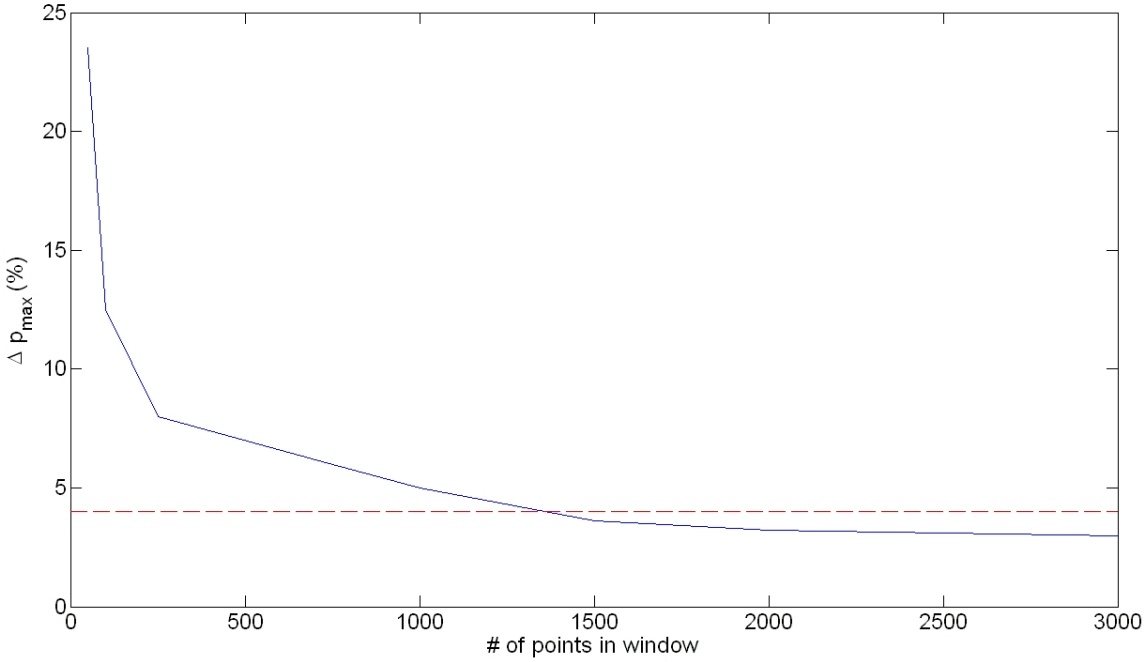

Fig. 22. The maximum probability deviations of moving windows which contain different number of points.

0.7 or 70%. Therefore, 0.7 is used as the fault detection threshold for the latter fault detector.

The Table I summarizes the parameters used for the two fault detectors.

3. Analysis of the Transient Results

Prior to analyzing the performance of the fault detectors, the transient result from each injected fault and special events are investigated in order to understand how the fault detectors work in some detail.

(a) Shopping Cart Fault

Fig. 23 shows the experimental result of the shopping cart fault using a segment of the workload profile that contains pint I in Fig. 21. The Fig. 23 (a) indicates

Table I. The parameters used for the fault detectors.

Parameter Name	Request Transition Matrix	Request File Distribution
Model Hit Threshold	50,000 points	48 hours
Detection Hit Threshold	1,500 points	N/A
Default Window Size	30 mins	30 mins
Detection Threshold	4%	70%

that the probability deviation from the shopping cart page to the shopping cart page crosses the fault detection threshold at 160 minutes. Therefore, one *abnormal circle* C_{ss} is detected, s is the ID of the shopping cart page. Since there is no other *abnormal direction* detected, the fault detector issues a fault alarm for this experiment at $t = 160$ minutes. Fig. 23 (b) shows the result of the fault detector using the request file distribution. The fault indicator crosses the fault detection threshold and the detector issues a fault alarm also at $t = 175$ minutes.

(b) Buy Request Fault

The result of the buy request fault using a segment of the workload profile that contains pint B is shown on the Fig. 24. Now request transition probability deviation from the buy request page to the shopping cart page (Fig. 24 (a)) and the one from the shopping cart page to the buy request page (Fig. 24 (b)) both cross the fault detection threshold at approximately 160-165 minutes. Therefore, An *abnormal circle* C_{sb} (s is the ID of the shopping cart page, b is the ID of the buy request page) is detected so the fault detector issue a fault. Fig. 24 (c) shows the result of the fault detector using the request file distribution. The fault indicator crosses the fault detection threshold and the detector issues a fault alarm also at $t = 159$ minutes.

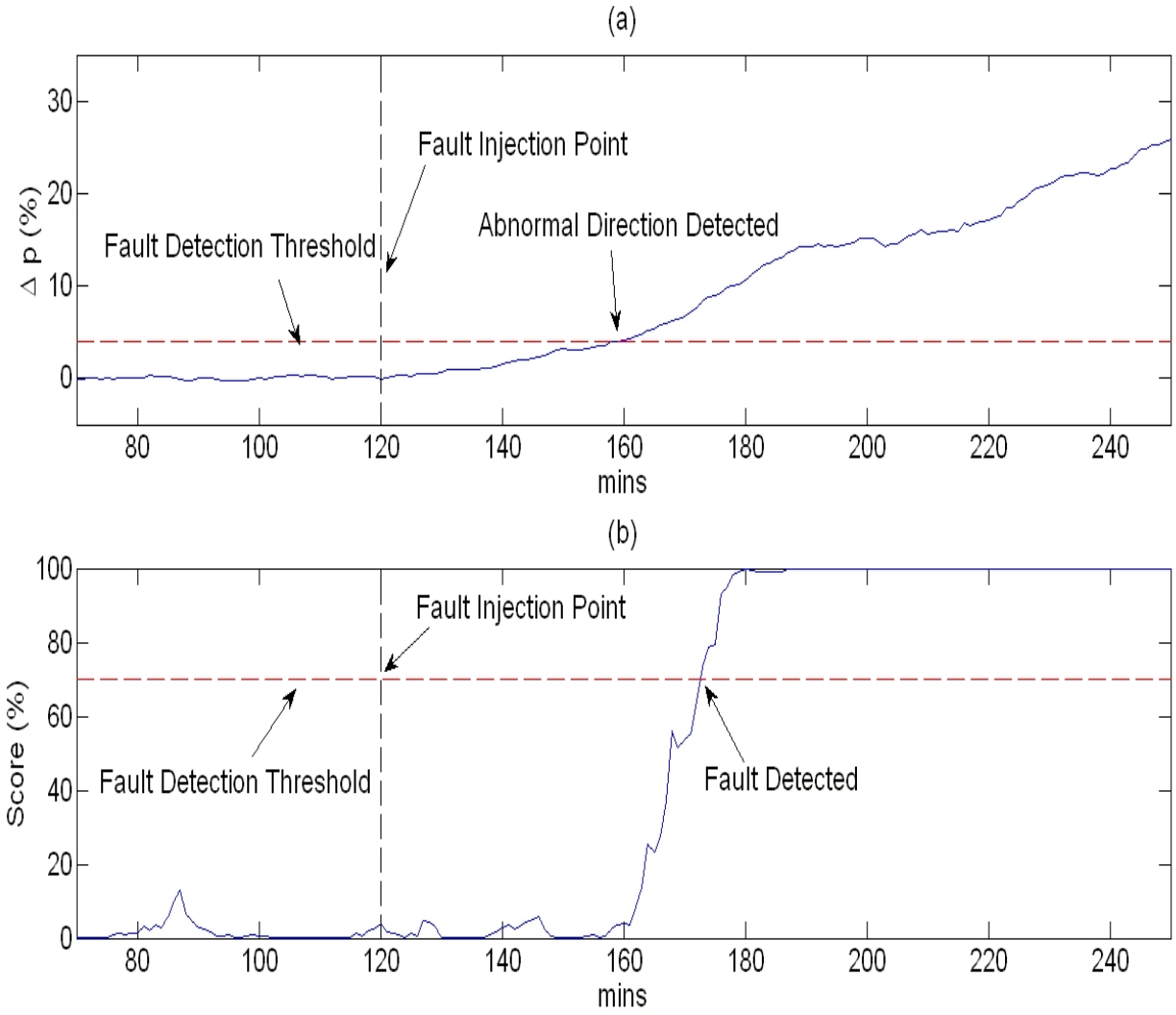

Fig. 23. Fault detection for the shopping cart fault: (a) Request transition probability deviation from the shopping cart page to the shopping cart page indicating a fault detected at 160 minutes. (b) Fault indicator of the fault detector using the request file distribution indicating a fault detected at 175 minutes.

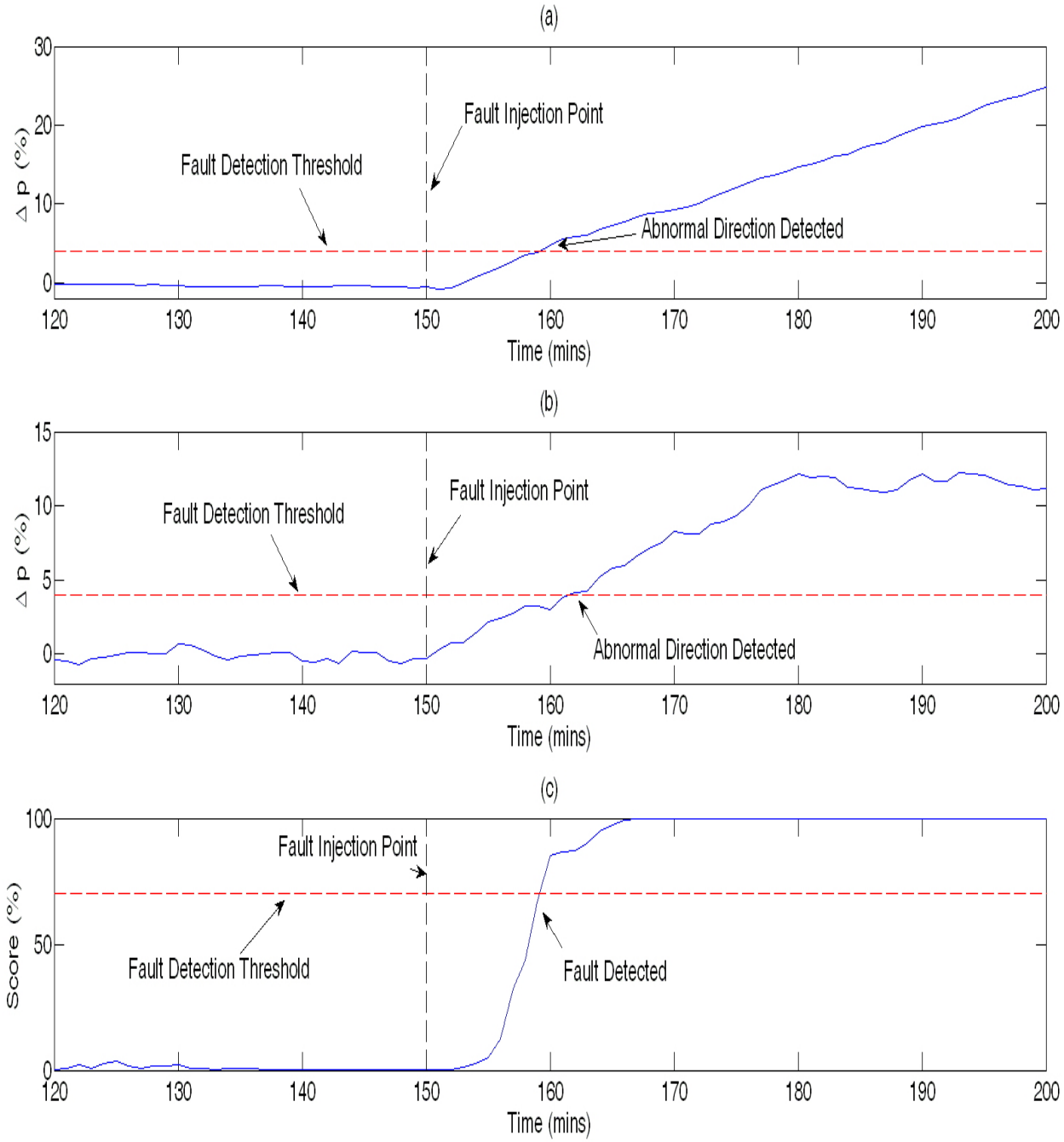

Fig. 24. Fault detection for the buy request fault: (a) Request transition probability deviation from the buy request page to the shopping cart page indicating an abnormal direction detected at 159 minutes. (b) Request transition probability deviation from the shopping cart page to the buy request page indicating an abnormal direction detected at 162 minutes. (c) Fault indicator of the fault detector using the request file distribution indicating a fault detected at 159 minutes.

(c) Search Fault

If end-users get unexpected search results, they might try the search several times and the request transition probability deviation from search result page to search result page rises. Fig. 25 (a) shows such deviation rise at t=160 minutes. An *abnormal circle* C_{ss} (s is the ID of the search result page) is detected and a fault alarm is issued by the fault detector. Fig. 25 (b) shows the result of the fault detector using the request file distribution. The fault indicator crosses the fault detection threshold and the detector issues a fault alarm also at $t = 191$ minutes.

(d) Product Detail Fault

Fig. 26 shows the result of the experiment of the product detail fault using a segment of the workload profile that contains pint I. The situation is similar to the one of the buy request fault. The transition probability deviation from the best seller page to the product detail page (Fig. 26 (a)) and the one from the product detail page to the best seller page (Fig. 26 (b)) both cross the fault detection threshold at 143 minutes and 156 minutes respectively. An *abnormal circle* C_{bp} (b is the ID of the best seller page, p is the ID of the product detail page) is detected and there is no any other *abnormal direction* detected. Therefore, a fault alarm is issued by the fault detector. Fig. 26 (c) shows the result of the fault detector using the request file distribution. The fault indicator crosses the fault detection threshold and the detector issues a fault alarm also at $t = 160$ minutes.

(e) New Released Item for Sale

Fig. 27 and Fig. 28 show an experimental result if the event of the new released item is injected. *abnormal directions* are detected in Fig. 27 (a), Fig. 27 (b), and Fig. 28

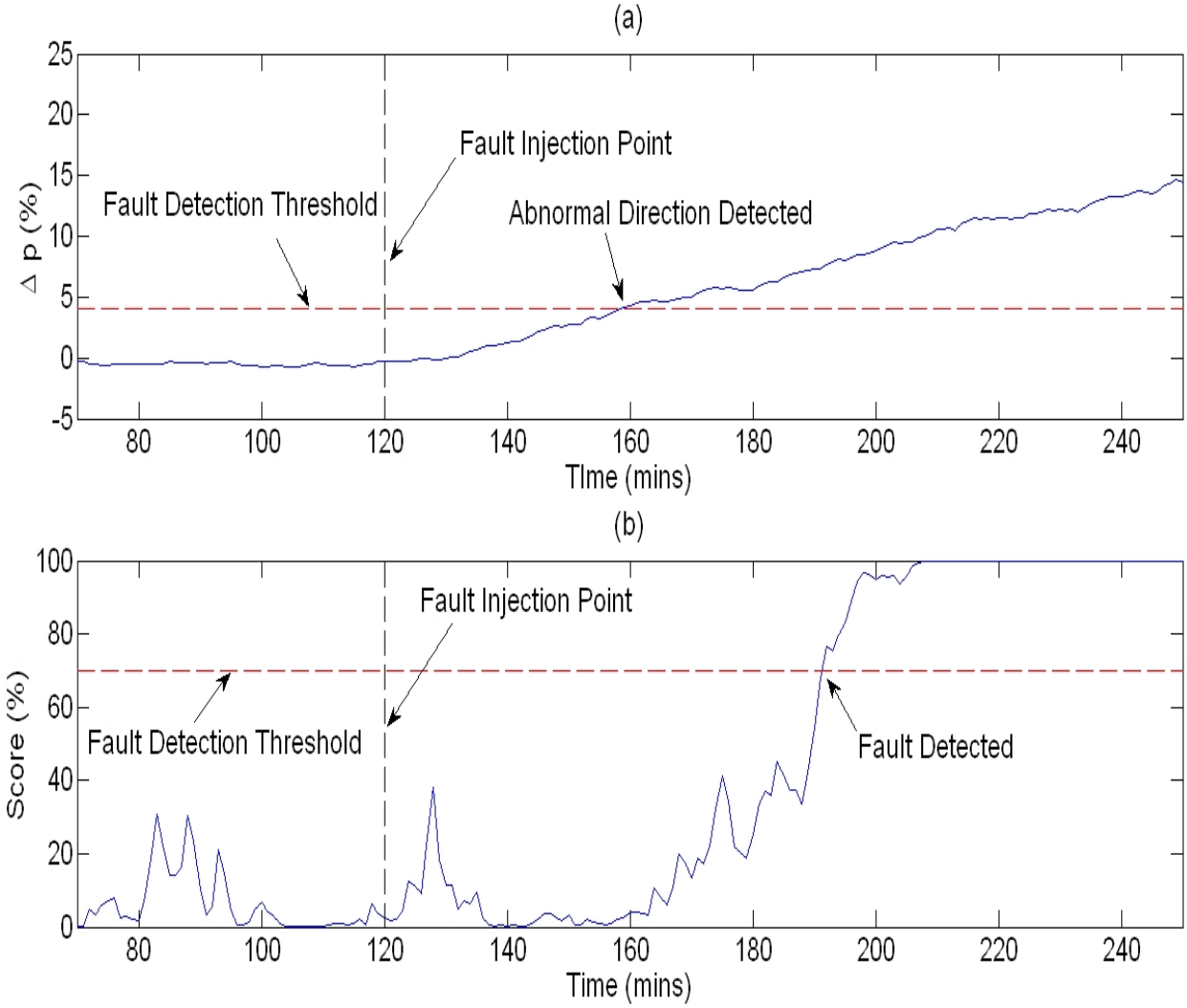

Fig. 25. Fault detection for the search fault: (a) Request transition probability deviation from the search result page to the search result page indicating an abnormal direction detected at 160 minutes. (b) Fault indicator of the fault detector using the request distribution indicating a fault detected at 191 minutes.

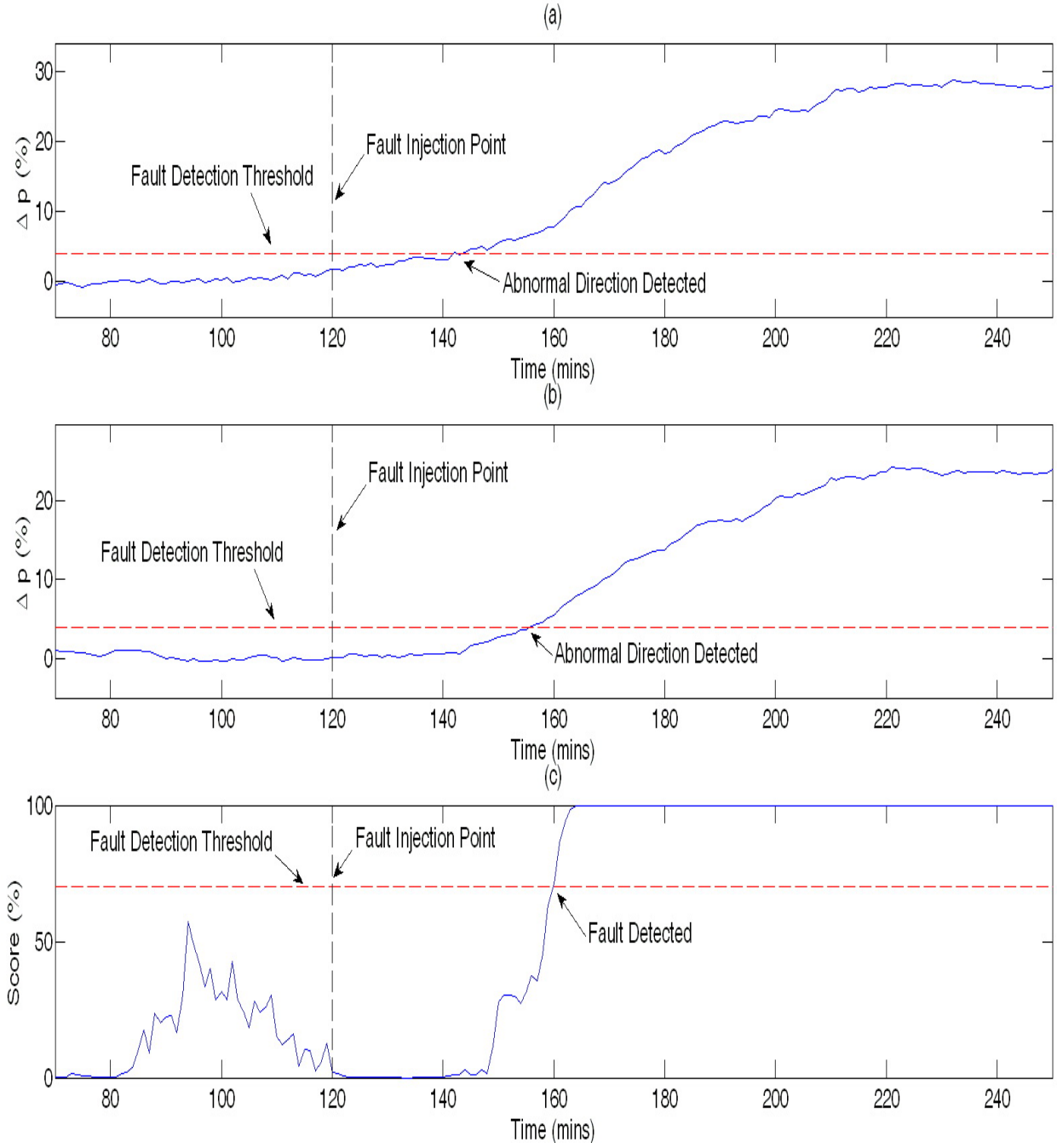

Fig. 26. Fault detection for the product detail fault: (a) Request transition probability deviation from the best seller page to the product detail page indicating an abnormal direction detected at 143 minutes. (b) Request transition probability deviation from the product detail page to the best seller page indicating an abnormal direction detected at 156 minutes. (c) Fault indicator of the fault detector using the request distribution indicating a fault detected at 160 minutes.

(a) at 140 minutes, 136 minutes, and 138 minutes respectively but since they don't form any *abnormal circle*, no fault alarm is issued by the fault detector. However, shown in Fig. 28 (b), for the fault detector using the request distribution, its fault indicator crosses the fault detection threshold at 135 minutes so that a fault alarm is issued. This causes a false alarm.

(f) Clearance Sale

The result of an experiment of the clearance sale is shown on the Fig. 29 and Fig. 30. From the Fig. 29 (b) and the Fig. 30 (a), an *abnormal circle* (C_{cp}, c is the ID of the catalog page and p is the ID of the product detail page) is formed at 142 minutes. However, shown on the Fig. 29 (a), there is an *abnormal direction* (D_{hc}, h is the ID of the home page) toward this *abnormal circle* detected at 139 minutes. Therefore, the fault detector using the request transition matrix does not issue a fault for this case. But the fault alarm is issued by the detector using the request file distribution since the fault indicator of the detector crosses the threshold at 139 minutes on the Fig. 30 (b).

(g) Best Seller Items on Sale

Fig. 31 and Fig. 32 show the experimental result of the event for the best seller items on sale. The situation is similar to the one of the event of clearance sale. According to the Fig. 31 (b) and Fig. 32 (a), an *abnormal circle* is detected approximately at 128 minutes but from the Fig 31 (a), an *abnormal direction* toward this circle is also detected at 126 minutes. Therefore, the fault alarm is not issued by the fault detector using the request transition matrix. However, the fault detector using request file distribution still issues a fault alarm at 128 minutes due to the high fault

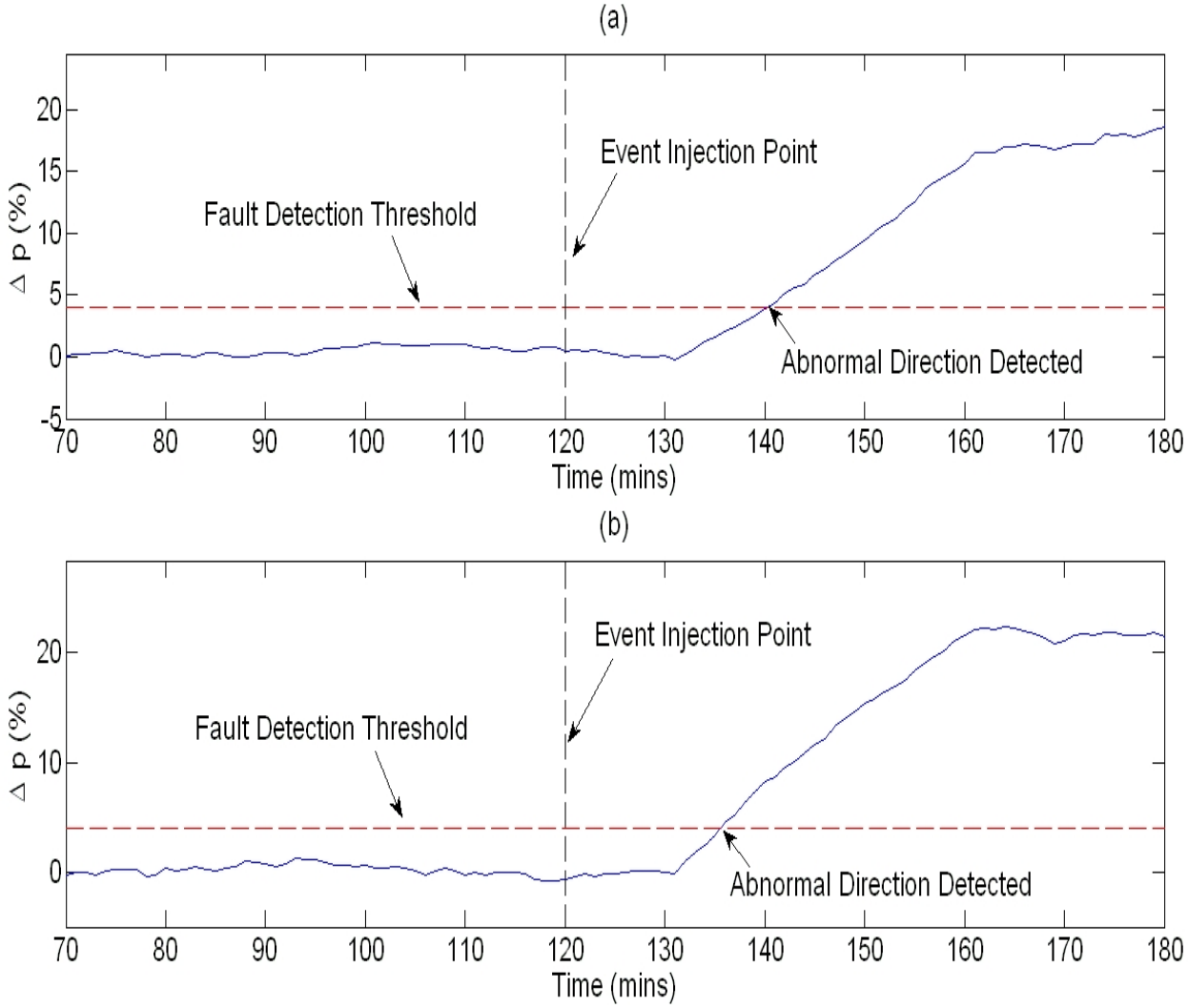

Fig. 27. The result for new released item event: (a) Request transition probability deviation from the home page to the new item page indicating an abnormal direction detected at 140 minutes. (b) Request transition probability deviation from the new item page to the product detail page indicating an abnormal direction detected at 136 minutes.

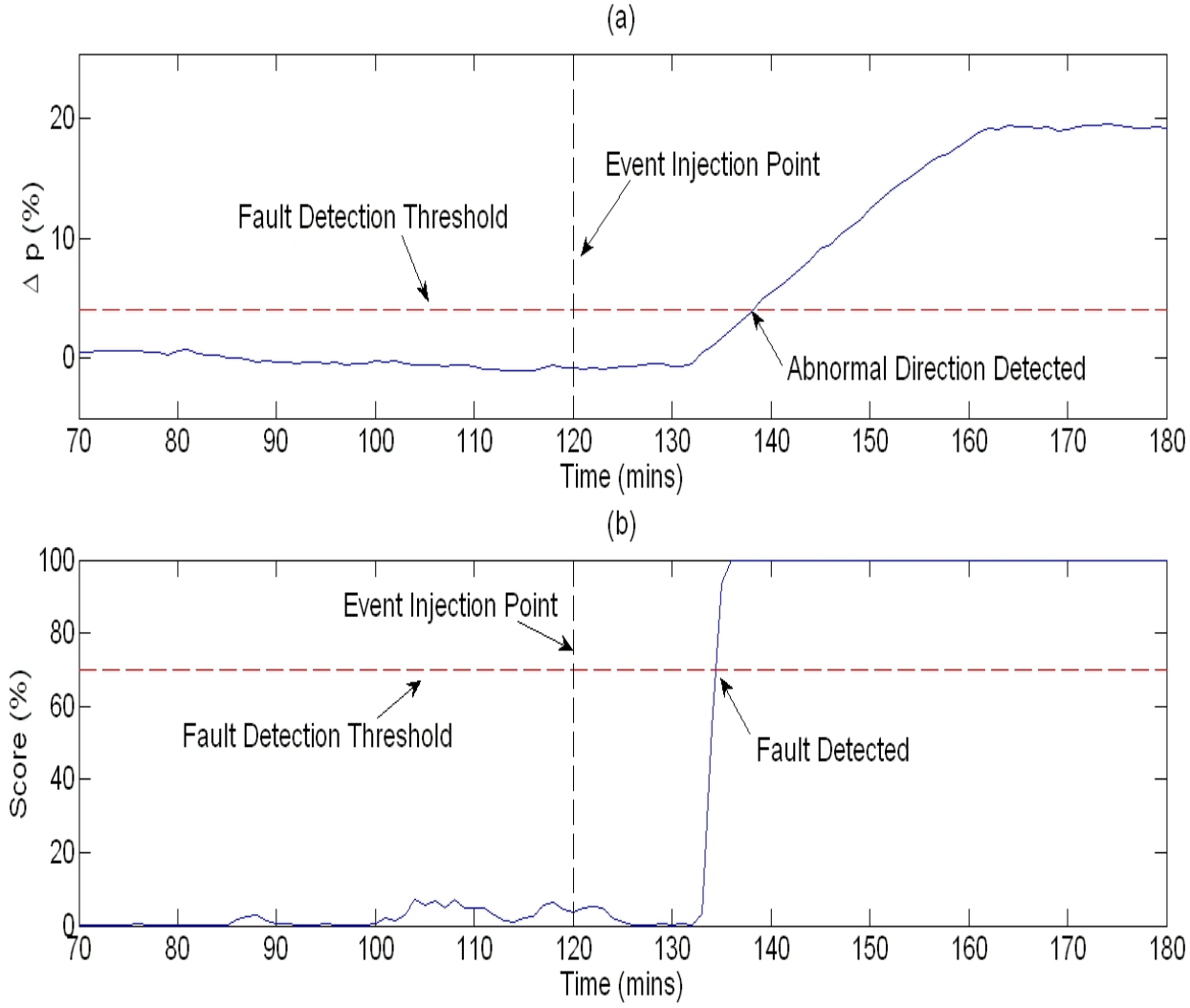

Fig. 28. The result for new released item event: (a) Request transition probability deviation from the product detail page to the shopping cart page indicating an abnormal direction detected at 138 minutes. (b) The fault indicator of the fault detector using the request file distribution indicating a fault detected at 135 minutes.

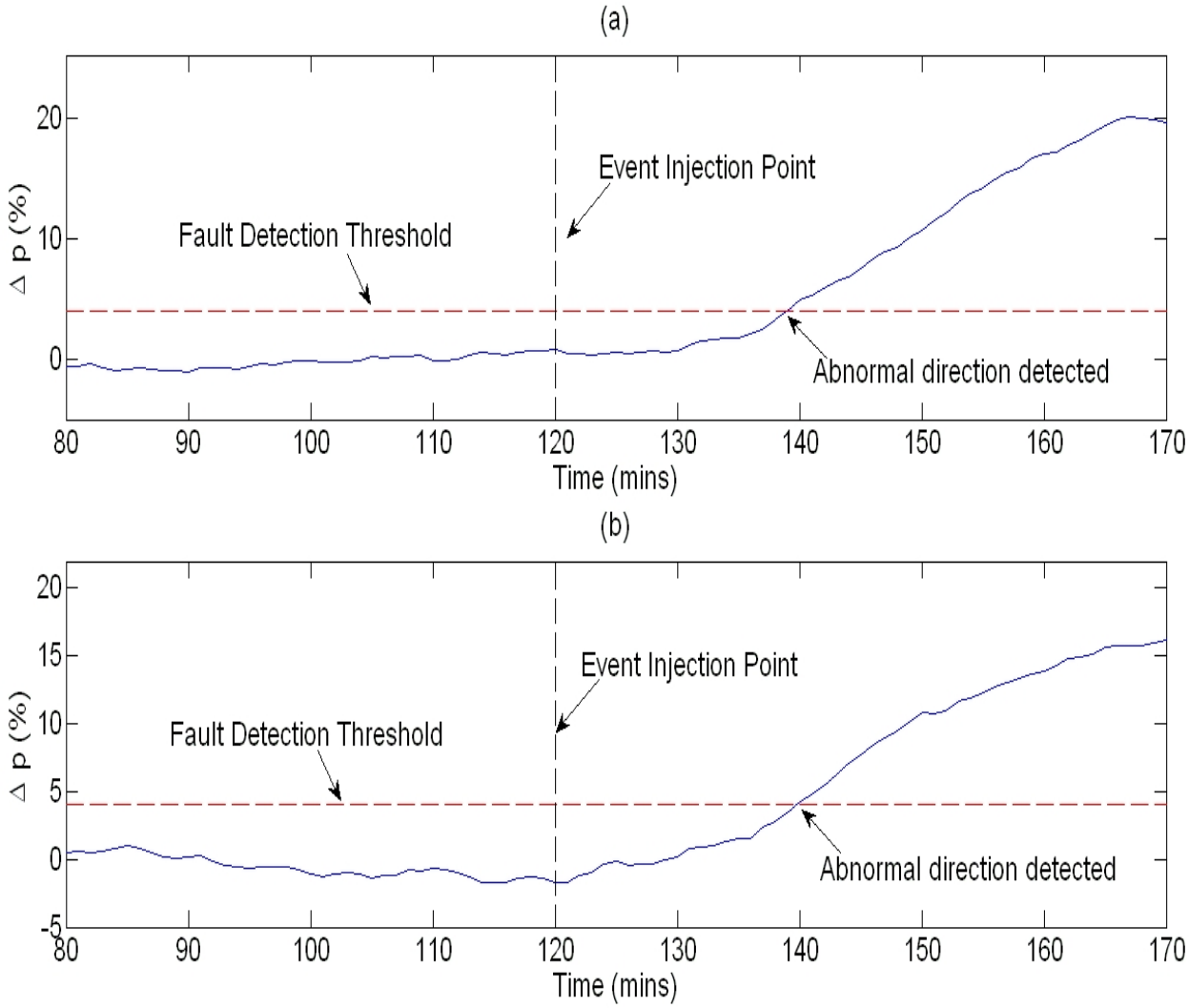

Fig. 29. The result for the event of clearance sale: (a) Request transition probability deviation from the home page to the catalog page indicating abnormal direction detected at 139 minutes. (b) Request transition probability deviation from the catalog page to the product detail page indicating abnormal direction detected at 140 minutes.

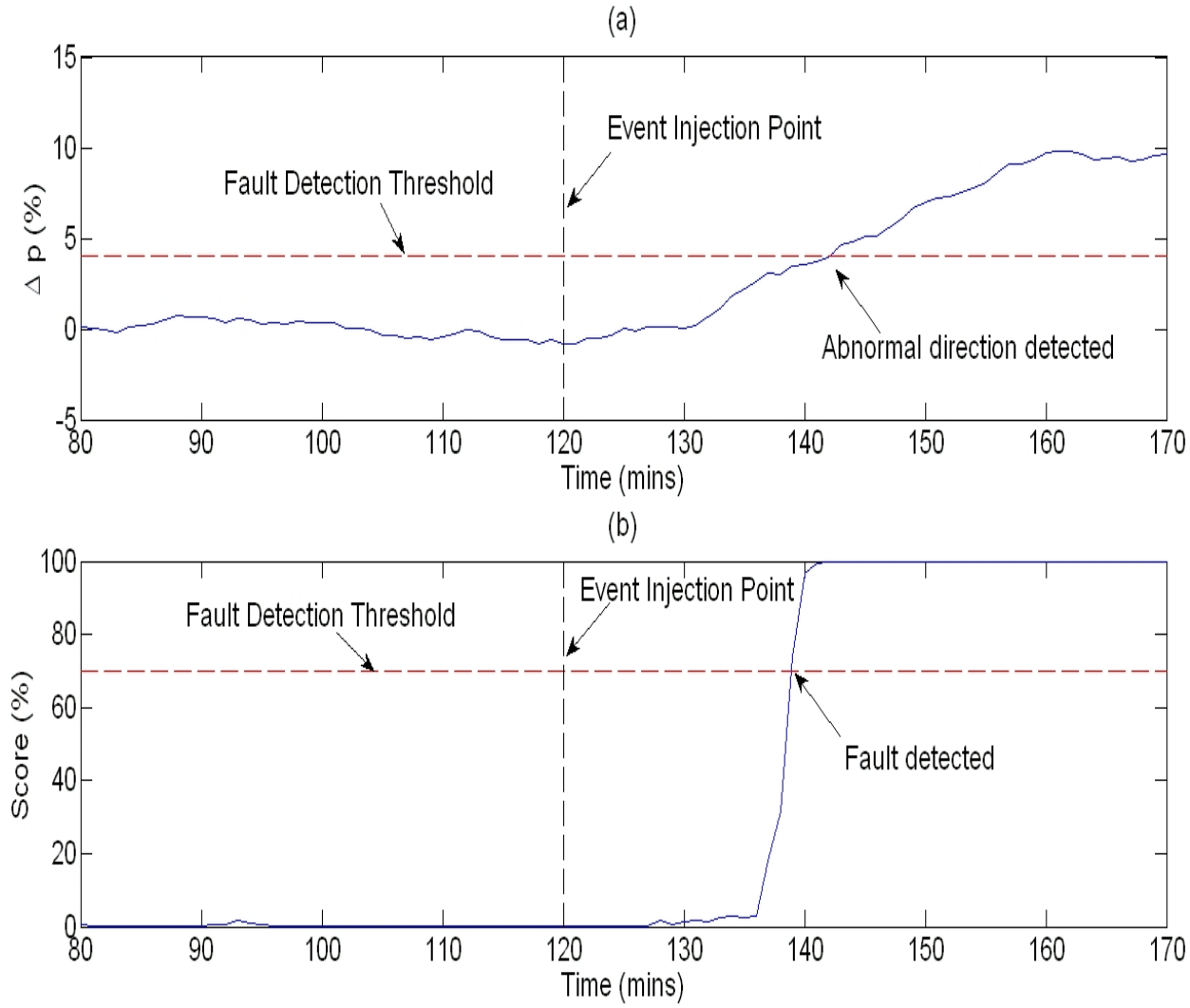

Fig. 30. The result for the event of clearance sale: (a) Request transition probability deviation from the product detail page to the catalog page indicating that an abnormal direction detected at 142 minutes. (b) The fault indicator of the fault detector using the request file distribution indicating a fault detected at 139 minutes.

indicator (shown in the Fig. 32 (b)).

From the experimental results of these four injected faults and three injected special events, it is clear to see that both the fault detector using the request transition matrix and the one using the request file distribution are able to detect all the four injected faults. However, only the former detector doesn't issue faults for the three special events. The latter detector still considers these special events as faults.

4. Performance Comparison Based on Detection Time

As mentioned in the previous section, the first method to evaluate the performance of the fault detectors is through the detection time. For each fault, experiments are conducted for nine different injected points, as shown on Fig. 21, so that the impact of different workloads on the detection time can be investigated. There are five experiments for each injected point in order to get the statistically significant results. The detection time of the two detectors for each fault investigated is discussed in details.

(a) Shopping Cart Fault

Fig. 33 shows the detection times under different workloads, i.e. varying number of end-users, for the shopping cart fault. This figure not only indicates the average detection time, square marker, but also the range of detection times at a certain workload level. It is clear that the detector using the request transition matrix can detect the shopping cart fault earlier than the one using the request file distribution, especially under a lighter workload. For example, under the workload with 50 end-users, the average detection time, 33 minutes, of the former detector is only 53% of the average detection time, 62 minutes, of the latter detector. Though Fig. 33

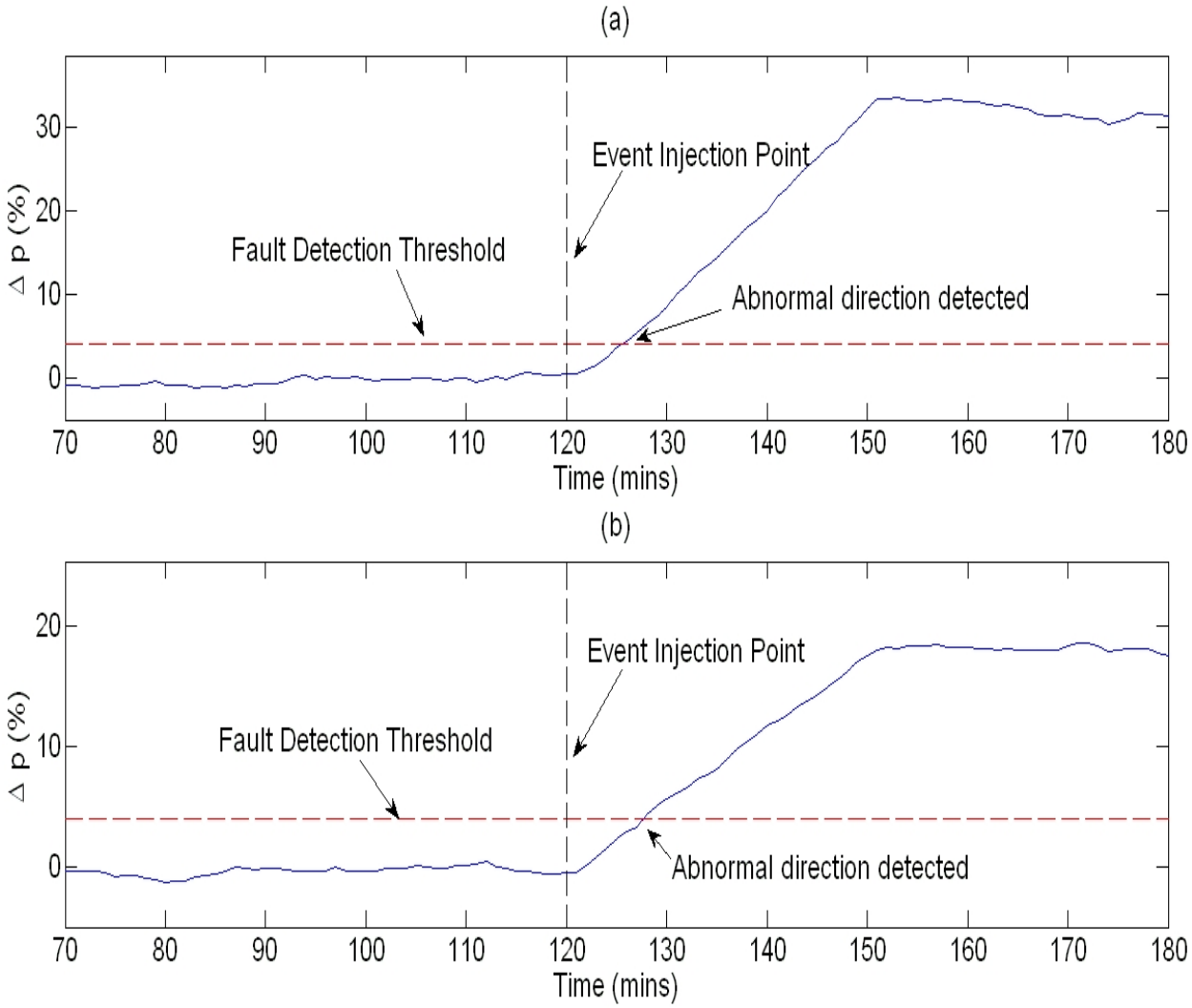

Fig. 31. The result of the event of best seller items on sale : (a) Request transition probability deviation from the best seller page to the product detail page indicating an abnormal direction detected at 126 minutes. (b) Request transition probability deviation from the product detail page to the shopping cart page indicating an abnormal direction detected at 128 minutes.

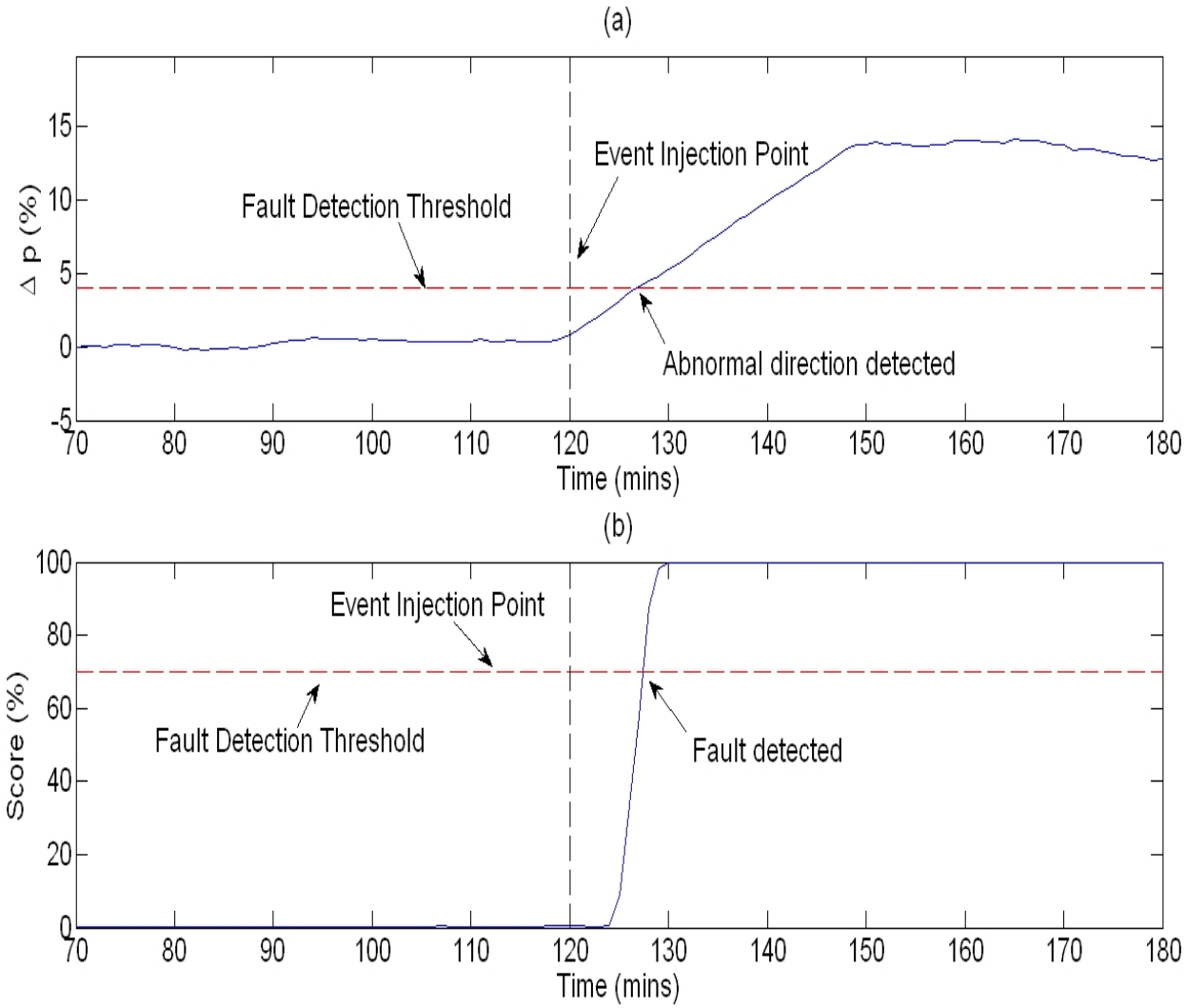

Fig. 32. The result of the event of best seller items on sale : (a) Request transition probability deviation from the shopping cart page to the product detail page indicating an abnormal direction detected at 127 minutes. (b) The fault indicator of the fault detector using the request file distribution indicating a fault detected at 128 minutes.

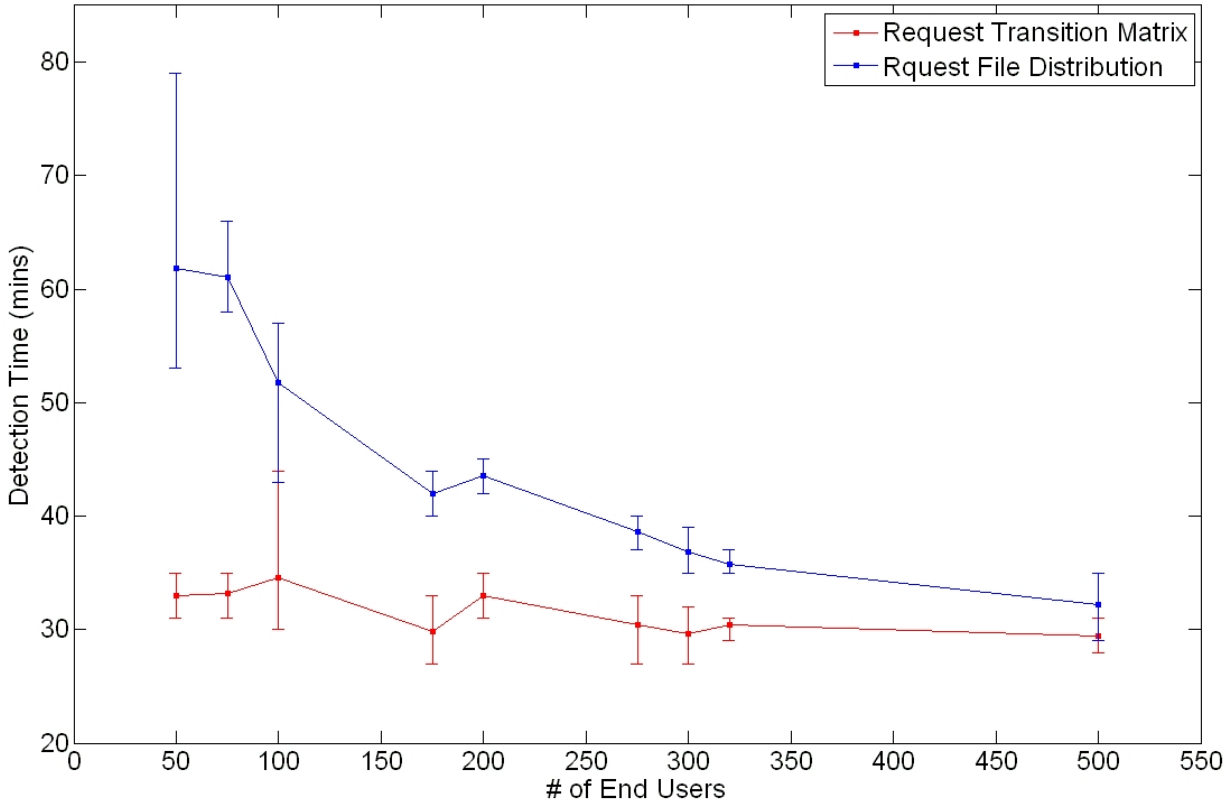

Fig. 33. Detection time for the shopping cart fault under different workloads.

shows that both detectors can have earlier detection if number of end-users increases, the detection time using the request file distribution is much more sensitive to the workload than the one of the fault detector using the request transition matrix. For the fault detector using the request file distribution, the percentage change of the detection time from the workload with 50 end-users to the one with 500 end-users is 48% while for the detector using the request transition matrix, it is only 11%. The overall average detection time of the fault detector using the transition matrix is 32 minutes whereas for the fault detector using the request file distribution is 45 minutes.

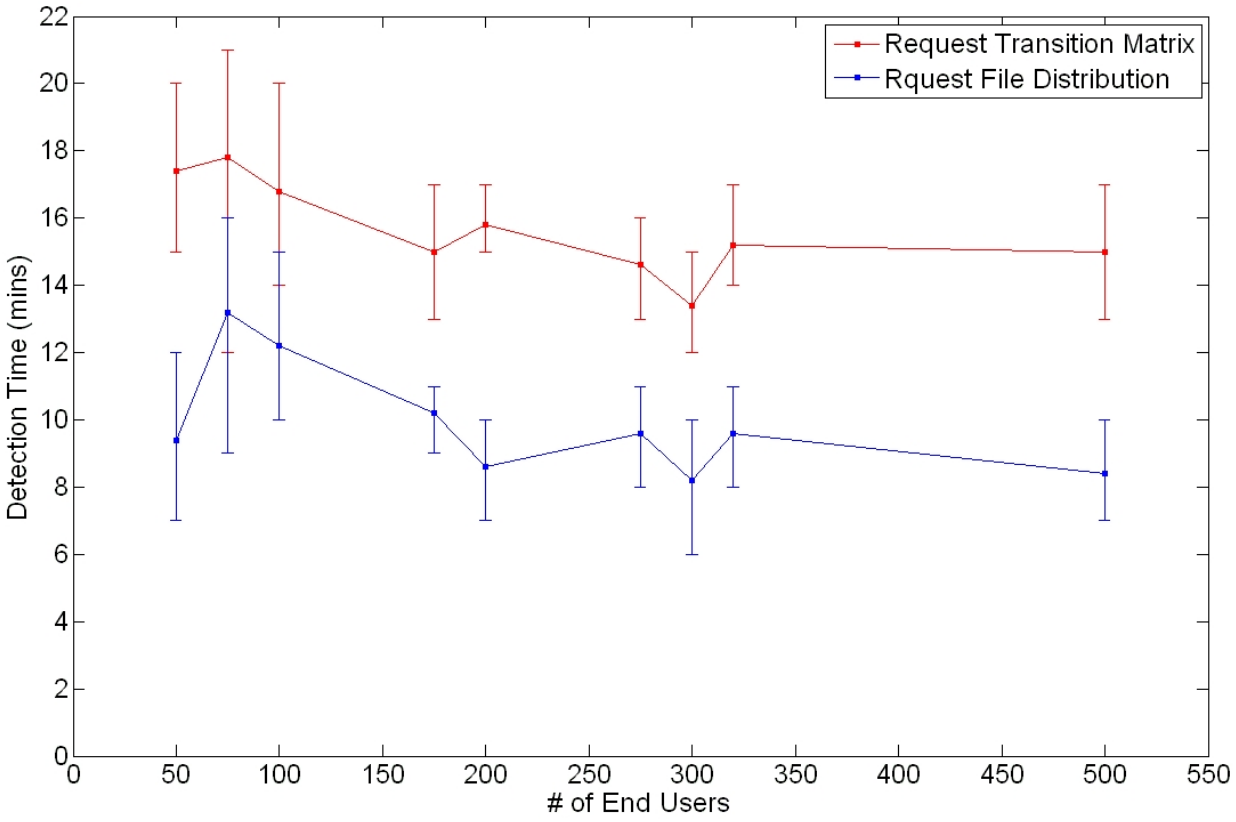

Fig. 34. Detection time for the buy request fault under different workloads.

(b) Buy Request Fault

Fig. 34 shows the detection times under different workloads for the buy request fault. It is clear form this figure that the fault detector using the request file distribution has earlier detection than using the request transition matrix for this type of fault. Also compared with the detection of the shopping cart fault, the detection time of the fault detector using the request file distribution is not as sensitive to workloads for this fault. The overall average detection time for the detector using the request transition matrix for this fault is 15 minutes whereas for the one using the request file distribution is 10 minutes.

(c) Search Fault

Fig. 35 shows the detection times under different workloads for the search fault. For this type of fault, the detection time using the request file distribution is still much more sensitive to the workload than the fault detector using the request transition matrix. For the fault detector using the request file distribution, the percentage change of the detection time from the workload with 50 end-users to the one with 500 end-users is 48% whereas for the detector using the request transition matrix is only 11%. Overall, the fault detector using the request transition matrix has better performance than the one using the request file distribution for this type of fault. However, the latter detector has slightly earlier detection if the number of end-users is more than 250. The overall average detection time for the detector using the request transition matrix and the one using the request file distribution is 46 minutes and 55 minutes respectively.

(d) Product Detail Fault

Fig. 36 shows the detection times under different workloads for the product detail fault. Overall, for this type of fault, the detector using the request file distribution has earlier detection than the one using the request transition matrix. But the latter detector has better performance if the number of the end-users is less than 100. The overall average detection time for the detector using the request transition matrix and the one using the request file distribution for this type of fault is 34 minutes and 31 minutes respectively.

From the results of the detection time for these four injected faults, generally detection time of the fault detector using the request transition matrix is less sensitive to workload changes than the detector using the request file distribution. The

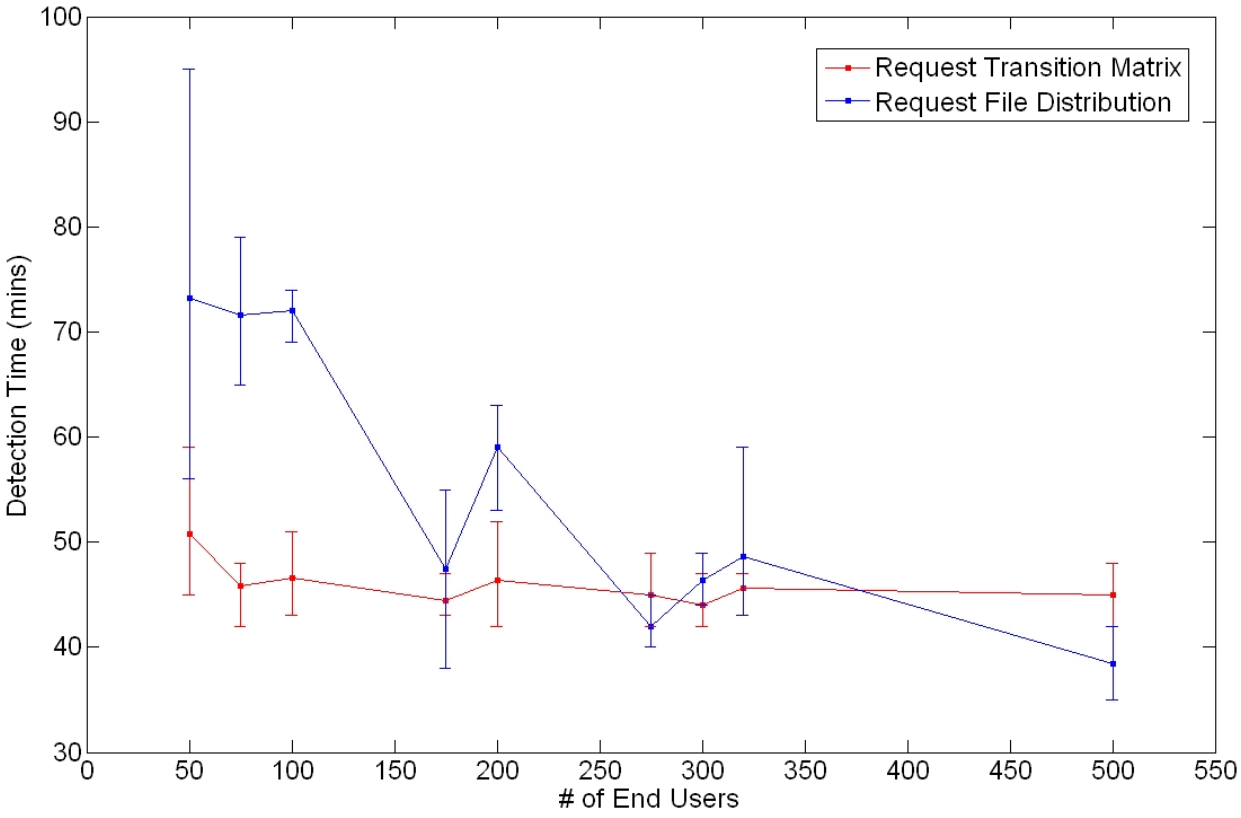

Fig. 35. Detection time for the search fault under different workloads.

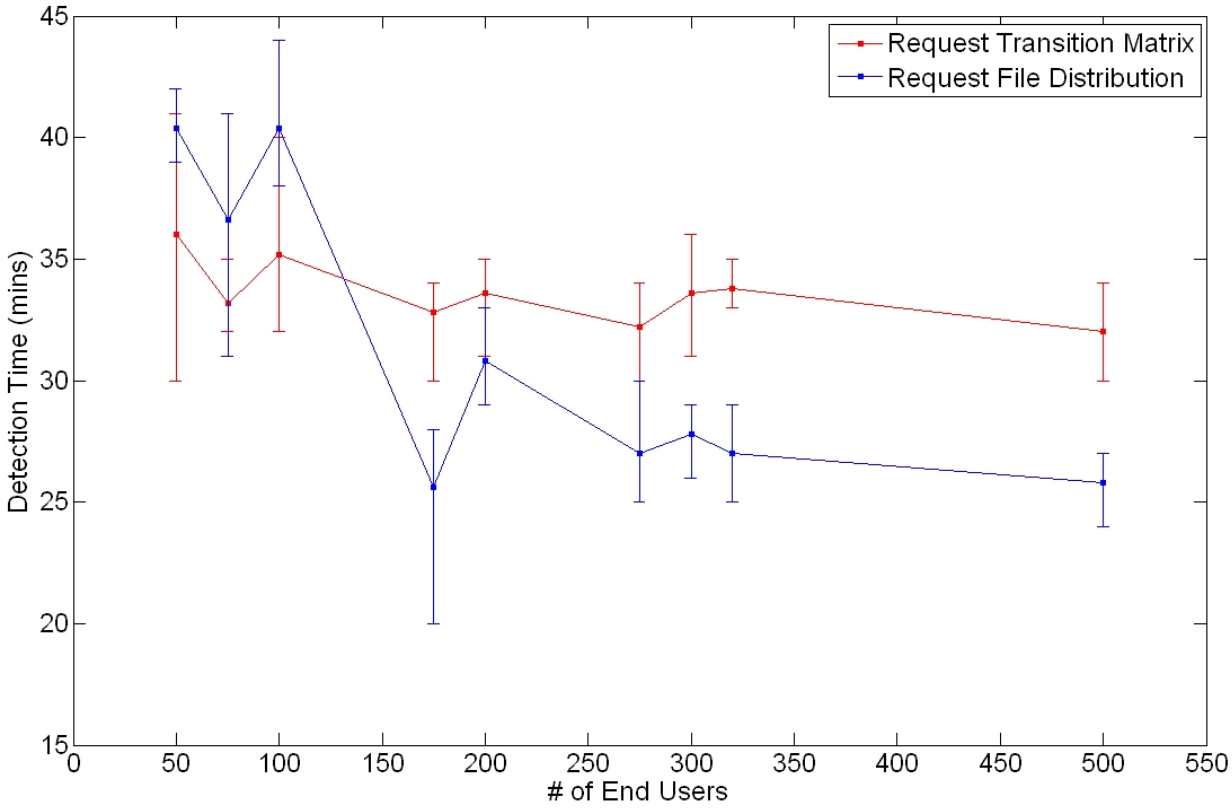

Fig. 36. Detection time for the product detail fault under different workloads.

former detector can provide better performance for two of these four types of faults investigated.

5. Performance Comparison Based on ROC Curve

The second approach for performance evaluation of detectors is through the ROC graphs. There are two ways to generate the ROC curves for both fault detectors. One is to fix the fault detection threshold and adjust the fault detection deadline. Another is to fix the fault detection deadline and change the fault detection threshold. The experiments for detection time evaluation, a total of 180 experiments, are used for the calculation of TP rates. In order to compute the FP rates, 100 experiments without any fault are used. These 100 experiments include 25 experiments with special events injected, 8 for the new released item for sale, 8 for the clearance sale, and 9 for the best seller items on sales, and 75 experiments without any special event.

The Fig. 37 shows the ROC curves for both detectors while the detection thresholds fixed at 4% for the detector using the request transition matrix and 70% for the one using the request file distribution, whereas the detection deadline is adjusted. It is clear to see that the detector using the request transition matrix has lower FP rate at a given TP rate. the detection deadline reaches 70 minutes, the FP rate is about 1%. For the detector using the request file distribution, the FP rate reaches 33% for the detection deadline as high as 60 minutes. The detector using the request file distribution has marginally better TP rate if the detection deadline is short but both detectors' TP rates are unacceptably low. The detector using the request transition matrix has higher TP rate the detection deadline is more than 30 minutes. Its TP rate reaches 100% if the detection deadline is 60 minutes, while the TP rate of the detector using the request file distribution is 94%, even if the detection deadline is at 70 minutes. Overall, according to this ROC graph, the detector using the request

transition matrix has better performance since its ROC curve is closer to the top-left corner of the ROC graph.

The Fig. 38 and Fig. 39 show the ROC curves for both the detectors if the detection thresholds are adjusted while the detection deadlines are set at 30 minutes and 50 minutes respectively. Usually the ROC curve of a detector approaches to the $(100\%, 100\%)$ point of the ROC graph while lowering its detection threshold, and approaches to the point $(0, 0)$ while raising its detection threshold. However, the ROC curves of the detector using the request transition matrix show different patterns. If the threshold is less than A certain value , 1% on both figures, the ROC curve of this detector approaches to the point $(0, 0)$, instead of point $(100\%, 100\%)$. This is because with a low threshold, though the chance to detect *abnormal circles* increases, the chance to detect *abnormal directions* toward the detected *abnormal circles* also increases. The detector doesn't issue a fault alarm if it detects an *abnormal circle* and an *abnormal direction* toward this circle because it considers this situation as a special event.

From both the Fig. 38 and the Fig. 39, it is clear that the detector using the request transition matrix can provide lower FP rate than the one using the request file distribution. The highest FP rates of the former detector are 12% and 16%, using the threshold 1%, from Fig. 38 and Fig. 39 respectively. This value is lower than the FP rate, 29% from the Fig. 38 and 32% from the Fig. 39, of the latter detector if using the default threshold, 70%. From Fig. 38, the TP rates of both the detectors are comparable. However, from the Fig. 39, the detector using the request transition matrix can provide better TP rate than the one using the request file distribution. The TP rate of the latter detector with low threshold 10% is 99%. The former detector can have comparable TP rate, 97% even using the default threshold 4%. Overall, according to Fig. 38 and Fig. 39, the detector using the request transition

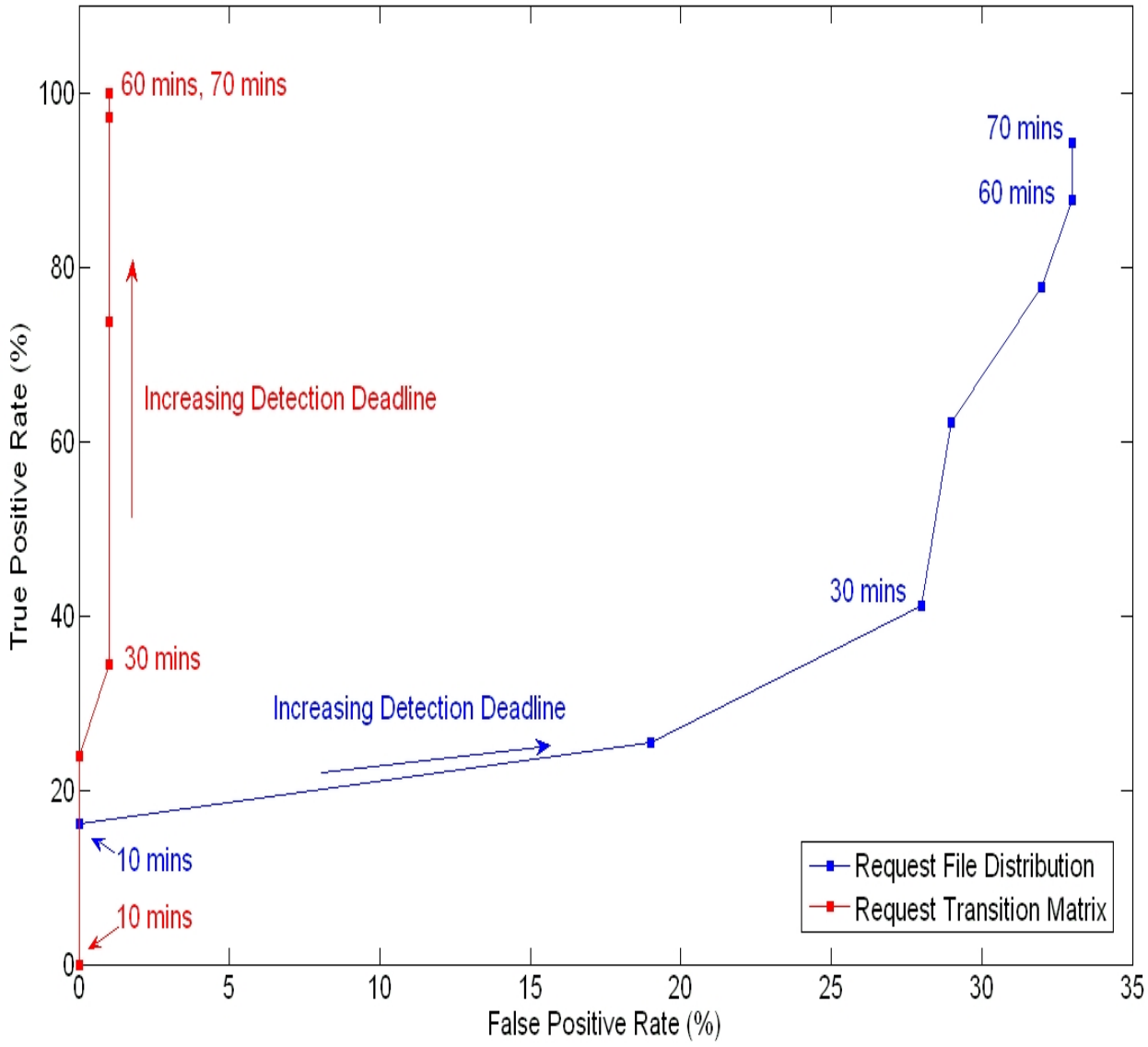

Fig. 37. The ROC graph for the two detectors with a fixed detection thresholds and variable fault detection deadline.

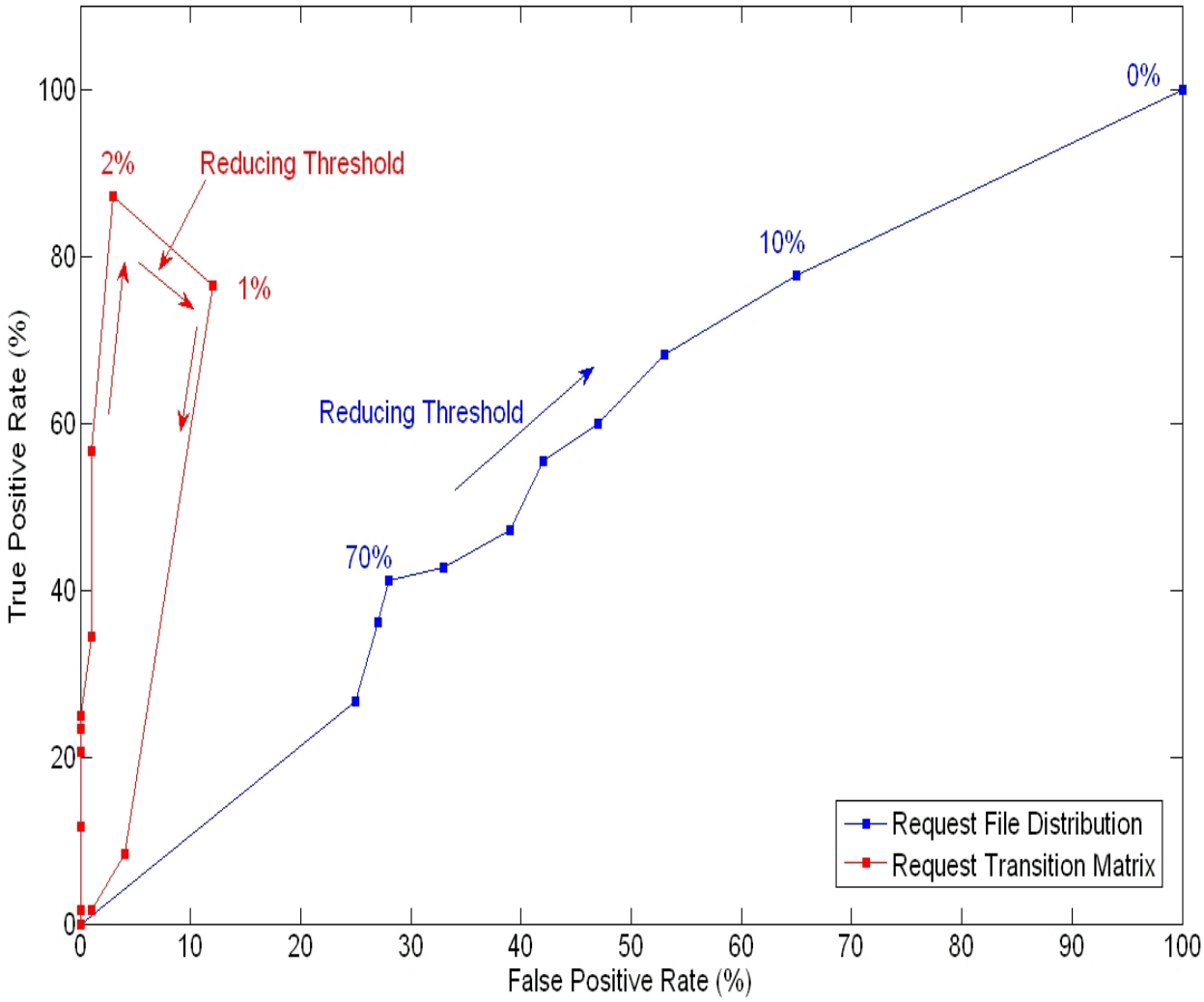

Fig. 38. The ROC graph for the two detectors with a fixed detection deadline, 30 minutes, and the variable detection threshold.

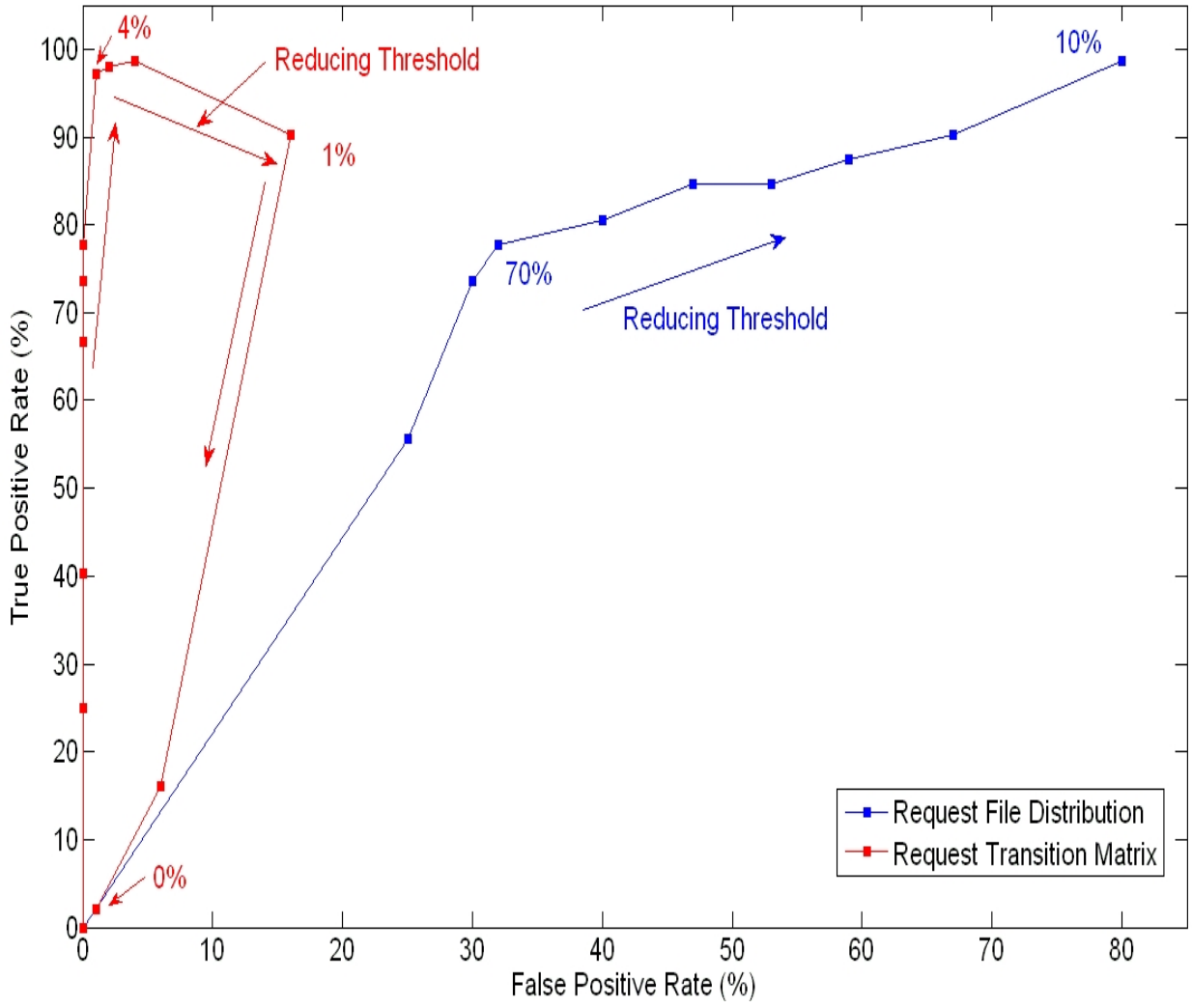

Fig. 39. The ROC graph for the two detectors with a fixed detection deadline, 50 minutes, and the variable detection threshold.

matrix is better than the one using the request file distribution, because the ROC curves of the former detector are closer to the top-left side than the ROC curves of the latter detector.

E. Summary and Discussion of the Off-line Testing

This chapter mainly discusses the setup of the testbed for the off-line testing and the off-line testing results for two different fault detectors: the detector using the request transition matrix and the one using the request file distribution. The architecture of the testbed for the off-line testing and its software environments and hardware specification are described in the first section. An emulated online bookstore following the TPC-W specification is setup as an E-commerce site under test. It contains a web server, an application server, and a database server and is able to provide all of the essential functions of a real online store. A client machine is setup as a workload generator and it produces EBs which emulate end-users' behaviors toward the online store.

The later sections introduce four different types of faults which are injected during the experiments and their possible causes. Three injected special events are introduced as well. With these four types of injected faults and three types of injected special events, the performance, i.e. the TP rate, the FP rate, and detection time, of the fault detectors can be evaluated. The next section describes two methods used to evaluate performance of the fault detectors. The first method is through the detection time of the detectors and the second method is through the ROC curves. In order to generate ROC curves for a fault detector, two approaches are used. The first one is to fix the detection threshold and vary the detection deadline, and the second one is to fix the detection deadline and vary the detection threshold.

In the following section, the experimental results of the off-line testing are discussed. First the parameters of the fault detectors are chosen and the workload profiles for the experiments are introduced so that the performance of the fault detectors under different workload levels is investigated. Then the transient experimental results for different faults and special events are studied. The results indicate that both the fault detector using the request transition matrix and the one using the request file distribution are able to detect the injected faults but only the former detector doesn't issue fault alarms for the three special events. For the performance evaluation and comparison, the detection time of the fault detectors for four different injected faults under different workload levels are investigated. The detection time of the fault detector using the request transition matrix is less sensitive to the workload changes than the one using the request file distribution and it has better performance for two, shopping cart fault and search fault, of the four types of faults. Even for other two faults, buy request fault and product detail fault, the difference in the average detection time for the two detectors is not significant. The Table II shows the average detection time of the two detectors under different faults. The differences in the detection time for these two detectors are 5 minutes and 3 minutes under the buy request fault and product detail fault respectively while they are 14 minutes and 9

Table II. The average detection time of two fault detectors for the four injected faults.

Fault type	Request transition matrix	Request file distribution
Shopping cart	31 mins	45 mins
Buy request	15 mins	10 mins
Search	46 mins	55 mins
Product detail	34 mins	31 mins

Table III. The performance metrics if the detection deadline is 30 minutes and the FP is 5%.

Detector	TP	FP	Precision	Accuracy
Request File Distribution	5%	5%	64%	37%
Request Transition Matrix	84%	5%	97%	88%

minutes for the shopping cart detail fault and the search fault, respectively. Finally, the ROC graphs for the two detectors are discussed for performance comparisons. Two methods, fixing the threshold and changing the detection deadline, fixing the detection deadline and changing the threshold, are used to generate a ROC curve for both detectors. According to the ROC graphs using these two methods, the detector using the request transition matrix provides better performance than the detector using the request file distribution because its ROC curves are closer to the top-left corner of the ROC graphs.

Based on Fig. 38 and Fig. 39, while choosing a proper fault detection threshold for the fault detectors to meet the desirable performance, it seems that the detector using the request transition matrix can provide better pairs of (TP, FP). While the detection deadline is set as 30 minutes, Table III shows the related performance matrices if the FP rate is set as 5% and Table IV shows the performance matrices if the TP rate is set as 86%. It is clear that from both tables, the detector using the request transition matrix can provide better performance. If the detection deadline is set as 50 minutes, the results are shown on the Table V and Table VI. From these two tables, the detector using the request transition matrix provides better performance as well.

Table IV. The performance metrics if the detection deadline is 30 minutes and the TP is 86%.

Detector	TP	FP	Precision	Accuracy
Request File Distribution	86%	80%	66%	62%
Request Transition Matrix	86%	3%	98%	90%

Table V. The performance metrics if the detection deadline is 50 minutes and the FP is 5%.

Detector	TP	FP	Precision	Accuracy
Request File Distribution	10%	5%	78%	40%
Request Transition Matrix	96%	5%	97%	96%

Table VI. The performance metrics if the detection deadline is 50 minutes and the TP is 95%.

Detector	TP	FP	Precision	Accuracy
Request File Distribution	95%	78%	69%	69%
Request Transition Matrix	95%	1%	99%	96%

CHAPTER V

REAL-TIME IMPLEMENTATION OF THE FAULT DETECTOR

A. Description of the Testbed and Differences from the Off-line Testbed

The testbed of the experiments for the online fault detector is similar to the off-line testbed. Fig. 40 depicts the structure of the testbed for the online fault detector. There are EBs running on the client machine to produce workloads for the emulated E-commerce site. The server machine has an emulated E-commerce site which contains a web server, an application server, and a database server. The only difference is that an additional computer is added as the monitoring site. The fault detection program runs on the monitoring site and it is responsible for retrieving the logged data from the E-commerce site and processing the data in order to determine whether a fault occurs.

The main reason for the fault detection program running on a different machine instead of the server machine is not to increase the load for the server machine. The E-commerce site on the server machine has to process requests from end-users and already is heavily-loaded. Using the structure of Fig. 40, the server machine just needs to send the logged data to the monitoring site and all the fault detection logics is executed on the monitoring site. There would not be much loading added to the server machine for fault detection. Moreover, usually a large E-commerce site is a cluster system and there are several server machines handling end-users' requests. It would be feasible to have a central monitoring site which receives logged data from all the server machines and handles the fault detection.

The operating system of the monitoring site is Microsoft Windows XP professional SP3. For the hardware configuration, it has an Intel Core 2 Quad Q9550

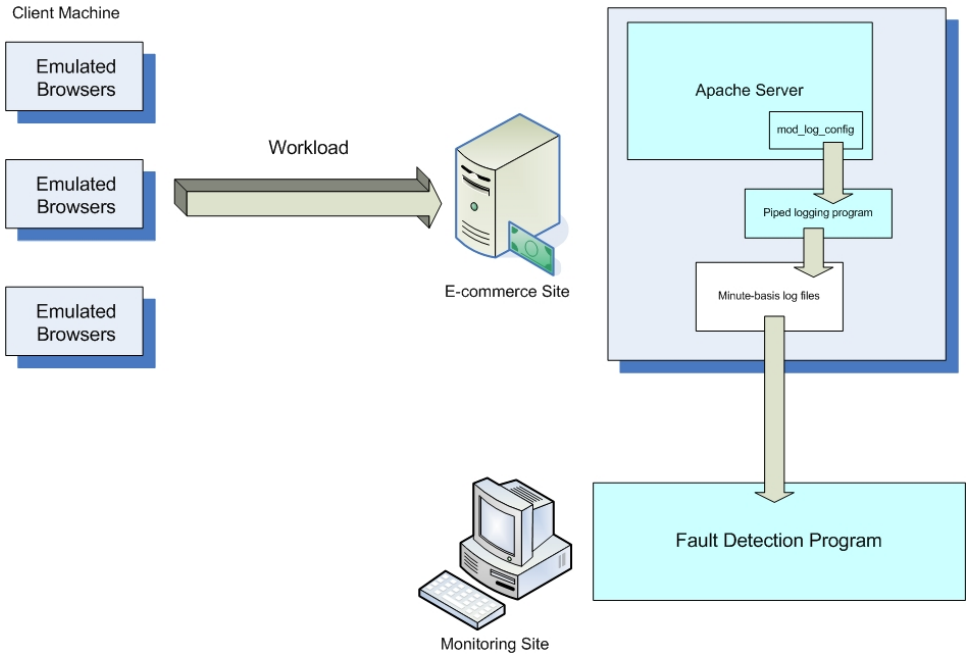

Fig. 40. The testbed of the online fault detector.

2.83GHz, with 4GB SDRAM, and a Seagate 80GB 7,200 rpm hard disk drive. It is networked to the server machine through a switched 1Gbps Ethernet LAN.

B. The Architecture of the Online Fault Detection System

Other than emulation software, there are two major programs for the online fault detection system. One is on the server machine and another is on the monitoring site. The main tasks of the program running on the server machine are to retrieve logged information from the web server and send them to the monitoring site. In order to get logged information from the web server, the pipeline technique provided by Linux is applied. A pipe [55] is an unidirectional communication channel between two programs. It works like a file pointer and just using the standard file I/O functions can establish the channel between two programs. Though this technique only provides an one-way communication, it is sufficient for the program on the server machine since

it just needs to receive the logged data from the web server and it doesn't have to receive a response back.

Another advantage of using the pipe technique is that *mod_log_config* [56], a module which is responsible for logging data for the Apache server, supports pipe communication. One just needs to specify which program should get data inside the Apache server configuration and *mod_log_config* sends the logged data to the program via a pipe. There is no need for modifying the source code of the web server. This makes the deployment of the fault detector fairly easy.

A simple program called *rotatelogs* [57] which is provided by the Apache software foundation is used to receive logged data from the web server. The main function of this program is to save the logged data into different log files based on the rotation time or the size of a log file. The rotation time is set to be one minute so that the program can create a new file for saving the logged data every minute and the detection program on the monitoring site can receive a log file for every minute for fault detection. Minor modification was made for the source code of the program *rotatelogs* in order to make it save the name of the latest file which finished storing the logged data to a specific file. The detection program on the monitoring site is able to determine which file is the latest log file based on the file name stored inside this specific file.

Fig. 41 shows the architecture of the detection program running on the monitoring site. Basically there are four different parts to this detection program: the file transfer function, the data interpretation, the local detection units, and the global detection unit. The task of the file transfer function is to receive log files from the server machine every minute. The file transfer is using the (secure shell) SSH file transfer protocol. This is the default Linux communication protocol and can provide an encrypted and secured channel for file transferring. After a log file is received by

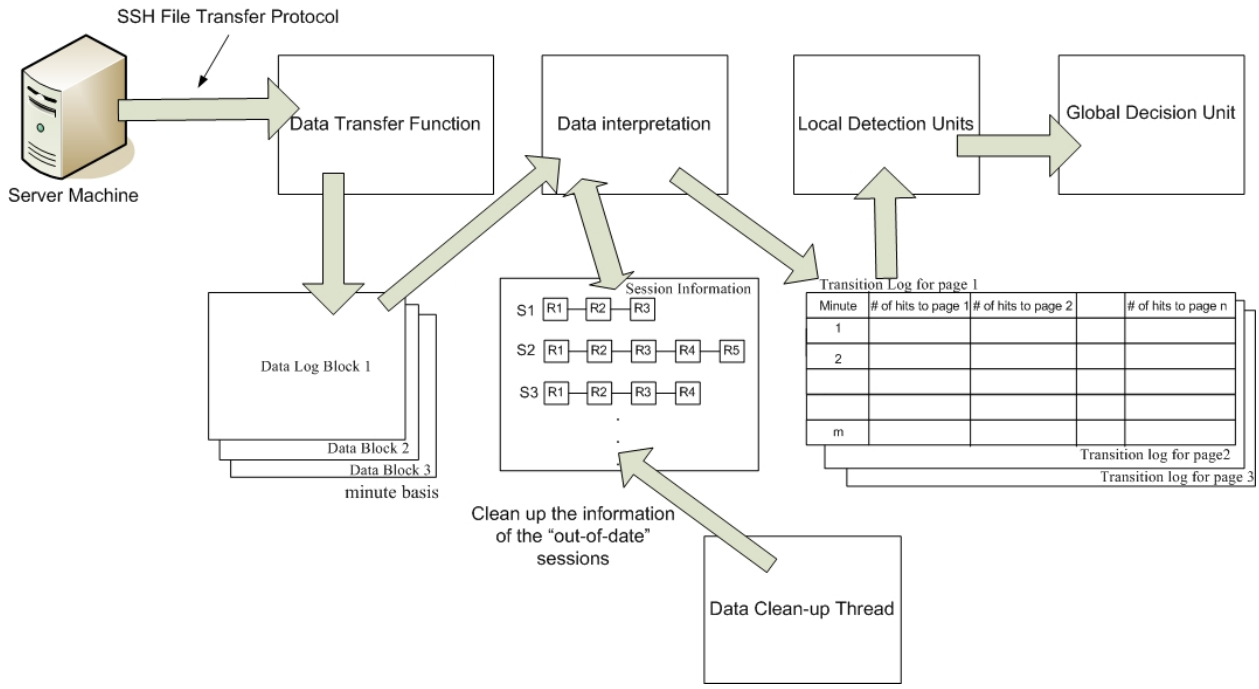

Fig. 41. The architecture of the detection program on the monitoring site.

the file transfer function, the data interpretation part starts to parse the data that are inside this log file. It filters out the records of the static objects, for example, image files, and records sets of consecutive requests for all end-user sessions. Based on the session information, it can have a transition log for each page which contains a row for each minute. Each row stores the number of hits within one minute from a certain page to each other page.

The last two parts of the monitoring program, the local detection units and the global detection unit have already been described in Chapter III. A local detection unit reads data from the transition logs, computes the transition probabilities to each page, for the page it is responsible for, and detects if there is any *abnormal direction*. The global detection unit collects information from all the local detection units and determines whether a fault alarm is issued. There is an additional thread called "data clean-up thread" and it periodically cleans up the information of the old sessions so

that the program would not occupy too much memory space.

C. Experimental Results

In order to conduct meaningful experiments for online testing for the fault detection system, the fault detector uses the identical sets of parameters with the off-line detector, shown on the Table I. The data for an 48-hour experiment under normal condition are used for the model building.

1. Results of the Injected Faults and Special Event

Three experiments, two with injected faults and one with injected special event, are investigated as follows:

(a) Shopping Cart Fault

Fig. 42 shows the experimental results of the shopping cart fault with the workload around 500 end-users. This figure shows that the transition probability deviation from the shopping cart page to the shopping cart page crosses the fault detection threshold at about 203 minutes. Therefore, one *abnormal circle* is detected and a fault is issued by the fault detector. The figure also shows both results from the online detector, the red line, and off-line detector, the blue line. The results are very similar but there is a short delay, less than 30 seconds, for the online detector due to the time it takes for the data transferring and data processing.

Fig. 43 shows the screen for the online detection program while it detects the shopping cart fault. It shows not only that the current status is faulty but also it point out which page has the problem. This achieves the goal of the fault detection and localization.

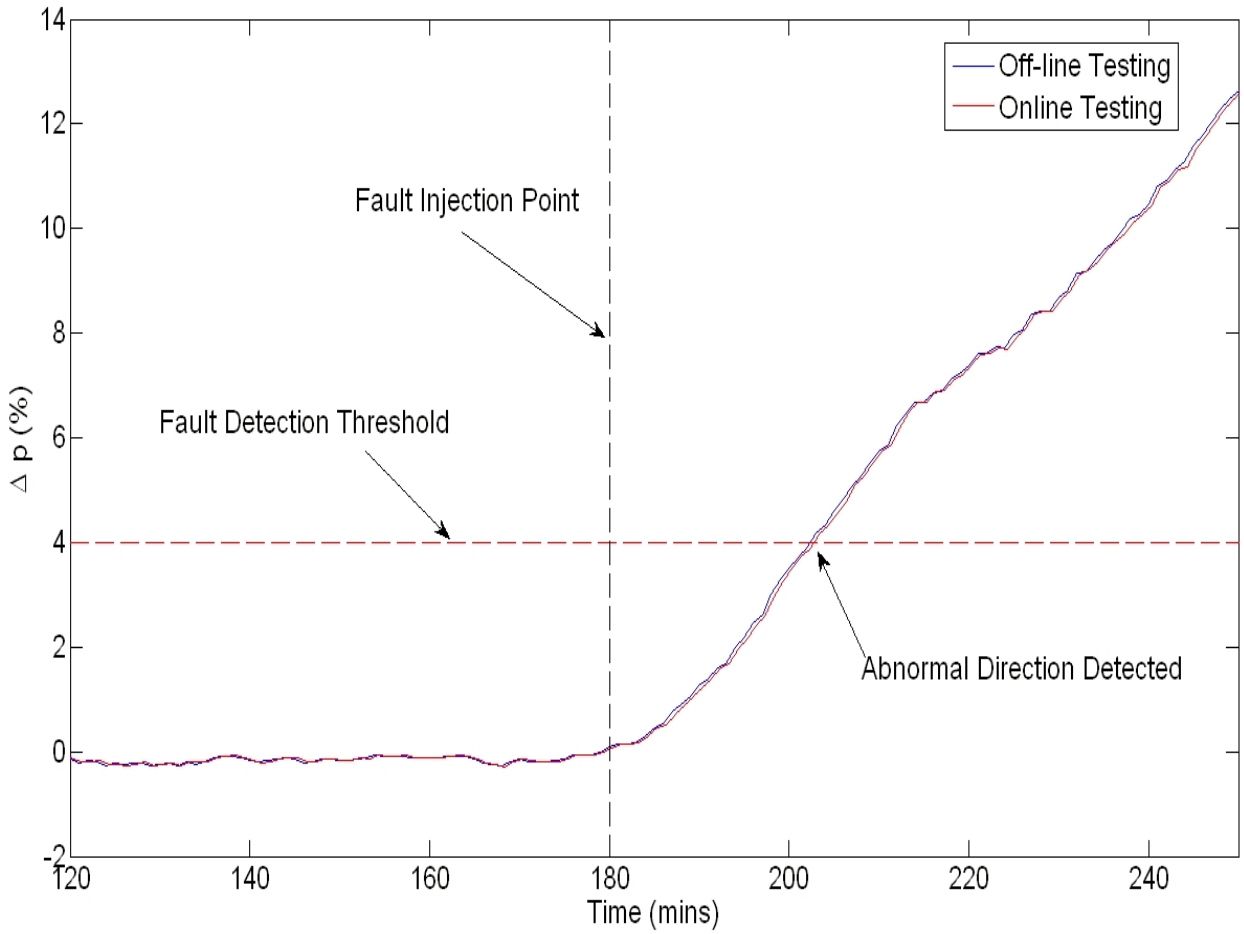

Fig. 42. The experimental result of the the off-line test and online testing for the shopping cart fault.

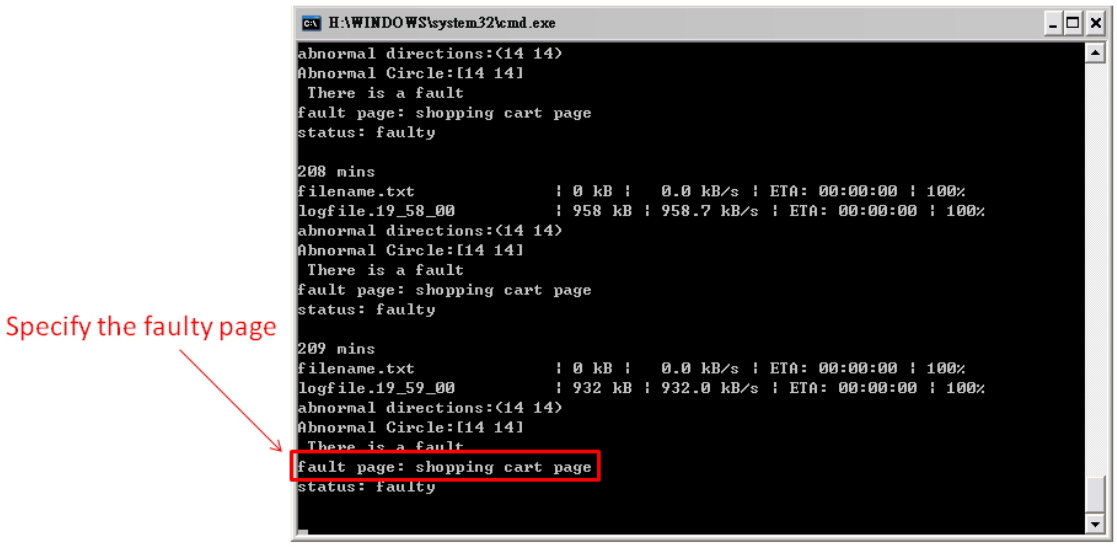

Fig. 43. The screen of the online fault detection program while it detects the shopping cart fault.

(b) Product Detail Fault

Fig. 44 shows the experimental result with the product detail fault at the workload around 300 end-users. Fig 44 (a) depicts that the transition probability deviation from the best seller page to the product detail page crosses the fault detection threshold at 203 minutes and the Fig 44 (b) shows that the transition probability deviation from the product detail page to the best seller page crosses the fault detection threshold at 211 minutes. Therefore, an *abnormal circle* is formed and a fault alarm is issued by the fault detector for this case.

(c) Best Seller Items on Sales

Fig. 45 shows the experimental results of the best seller items on sales under the workload around 300 end-users. Fig. 45 (b) and the Fig. 45 (c) shows that an *abnormal circle* is detected but Fig. 45 (a) shows that an *abnormal direction* toward

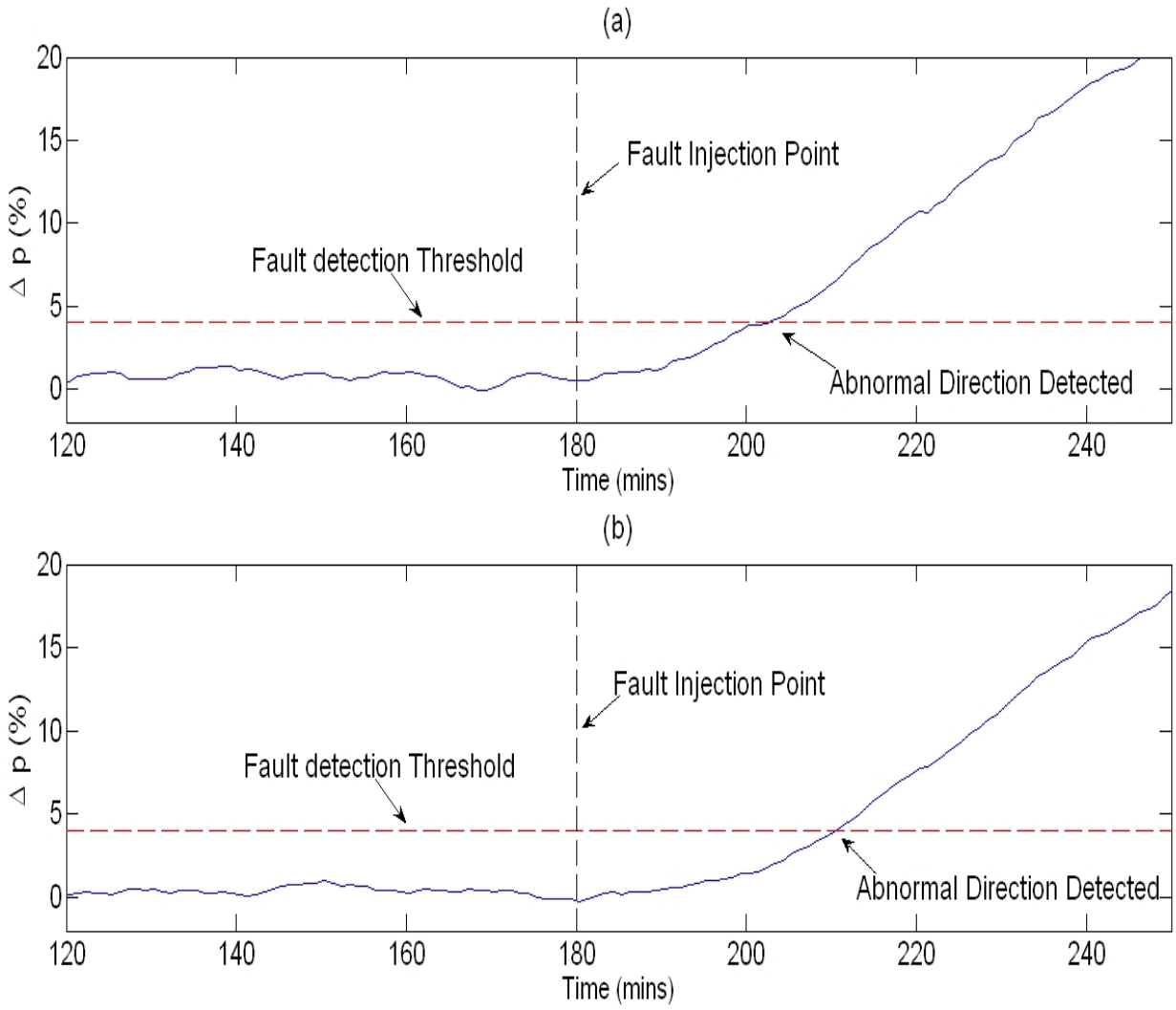

Fig. 44. The experimental results for the online testing for the product detail fault: (a) Request transition probability deviation from the best seller page to the product detail page. (b) Request transition probability deviation from the product detail page to the best seller page.

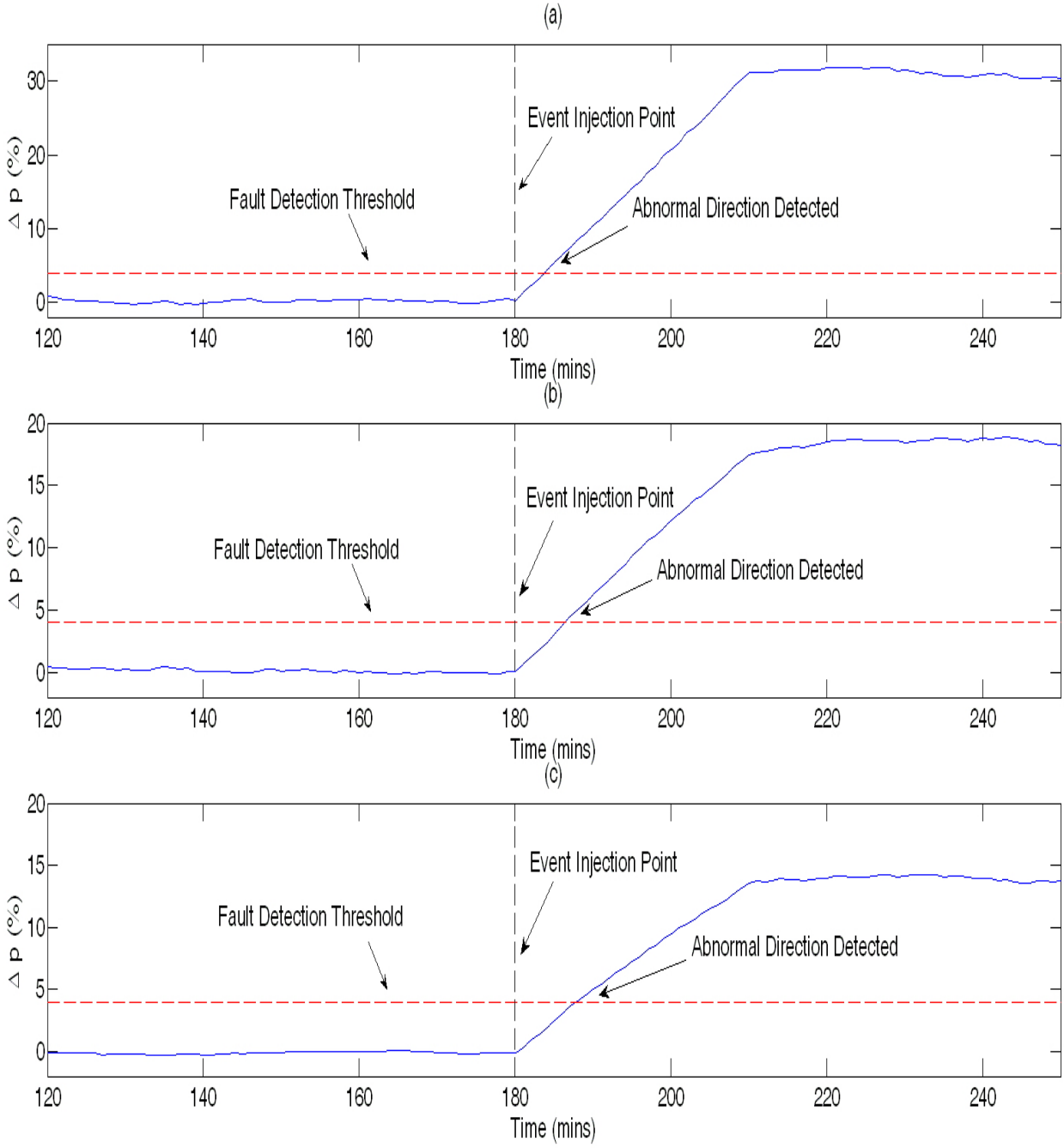

Fig. 45. The experimental results for the online testing for the best seller items on sales: (a) Request transition probability deviation from the best seller page to the product detail page. (b)Request transition probability deviation from the product detail page to the shopping cart page. (c)Request transition probability deviation from the shopping cart page to the product detail page.

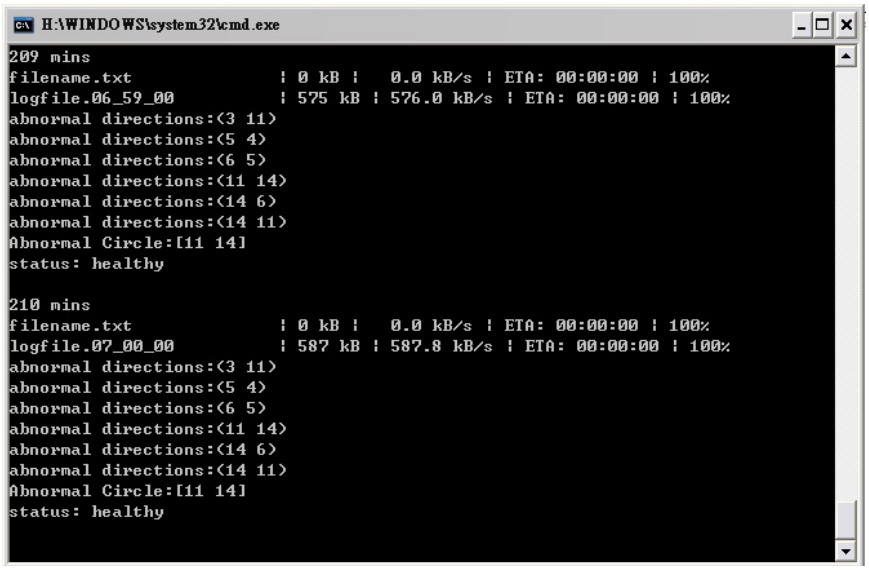

Fig. 46. The screen of the online detection program while it encounters the special event.

this *abnormal circle* is detected as well. No fault alarm is issued for this case.

Fig. 46 shows the screen for the online detection program while it encounters this special event. It shows that an *abnormal circle* ($C_{11,14}$) is detected but an *abnormal direction* ($D_{3,11}$) toward this circle is also detected. Therefore, the program still states that the status is healthy.

According to the results of the above three experiments, the online fault detector can work just as well as the off-line detector. It is able to detect faults and also distinguish between real faults and special events.

D. Summary and Discussion of the Online Testing

This chapter discussed the setup for the online detection system using the request transition matrix and its experimental results. The basic setup for the online detector is described in the first section. An additional machine called the monitoring site is added to the system and all of the detection algorithms are executed on this ma-

chine. The server machine just needs to send the logged data to the monitoring site. Therefore, deploying the detector would not increase loading of the server machine much.

The second section explains the architecture of the fault detection system. There are two major programs for the detection system. One is the piped logging program running on the server machine and another one is the detection program running on the monitoring site. Main tasks of the piped logging program are to get logged data from the web server and pack the data so that the detection program can get them for processing. Since the pipe technique is used for this program, no source code modification of the web server is needed and this simplified the deployment of the fault detection system. The main tasks of the detection program are to get data from the server machine, interpret the data, and perform the fault detection. All the detection algorithms and the detection processes of the program follows the methods described in Chapter III.

There are three experiments, two with injected faults and one with injected special event, discussed in the third section in order to verify the functions and performance of the online detector. From the results of these three experiments, the online detector can produce almost identical results to the ones from the off-line detector. It can distinguish between real faults and special events and if a fault occurs, it can figure out which page has the problem. The proposed online detector is really able to have all four characteristics mentioned in the first chapter.

CHAPTER VI

SUMMARY AND CONCLUSION

This chapter first summaries the research presented in the previous chapters. The research conclusions, its limitations, and the suggested future work are described in the last part of this chapter.

A. Summary of the Research

The first chapter briefly introduces the status of the current E-commerce industry. Though this industry is growing at incredible speed in the recent years, faults and unexpected downtime of E-commerce sites continue to cause companies to lose revenues. User-visible faults are introduced in this chapter and they are some of the major problems causing E-commerce sites to lose revenue and moreover, lose customer loyalty. Detecting user-visible faults sometimes is a difficult task since some of them do not produce any error messages in the log files of servers. The objective of this research work is to develop a detector which can detect user-visible faults automatically. This detector should have four important characteristics: early fault detection, false alarm reduction, easy deployment, and effective fault localization. Since none of the current detectors have all these four advantages,the fault detector using a newly defined concept, the request transition matrix, is proposed in order to achieve the research objective.

Chapter II provides an overview of the literature about the prior related research.. There are four parts of this chapters. The first part discusses the current research work regarding the performance model building, and performance improvement of web servers using control theory or scheduling techniques. The second part investigates the current research work related to fault detection in networked computer systems.

There are two main catalogs of works for this part: model-free fault detection and model-based fault detection. The third part investigates the most relevant literature for our work, the research done by the Recovery Oriented Computing (ROC) group which focuses on developing several novel techniques for building highly-dependable Internet services. This group has developed several fault detectors which can detect user-visible faults and one of these detectors (The χ^2 fault detector using request file distribution) is used for comparison with the fault detector developed by this research work. In order to produce realistic traffic profiles for the fault detection experiments, workload modelling which can emulate real world network traffic for E-commerce systems is discussed in the final part of this chapter.

Chapter III presents the algorithms for the proposed user-visible fault detector using the request transition matrix concept. First the basic concepts and two main approaches, model-free and model-based, for fault detection are discussed. Two performance metrics, true positive rate and false positive rate, and the ROC (not related to ROC group) curve are defined in order to evaluate the performance of a fault detector in this research work. The reasons the fault detector exhibits the four characteristics are explained. Since the request transition matrix can be computed from the log files, it is easily to deployed. The system can detect user-visible faults before it propagates to cause serious damage to performance or system crash so it has the feature of early fault detection. Two important terms, *abnormal direction* and *abnormal circle*, for this research work are then defined. Based on detected *abnormal circles* and *abnormal directions*, the fault detector not only can detect faults but also can exclude special events so it can reduce the false positive rate. Also according to *abnormal circles* it detects, it can determine which page has the fault so that it can exhibit the feature of fault localization. Finally, the architecture of the fault detection system is described. There are two major components to the fault detection system:

local detection units and a global detection unit. A local detection unit is responsible for adjusting the window size to collect sufficient number of hits to compute the transition probability for one page and for reporting the detected *abnormal directions* to the global detection unit. The global detection unit gathers all the information from all local detection units and determines whether or not to issue a fault alarm.

Chapter IV discusses the setup for the fault detection experiments and investigates the experimental results for the two fault detectors: the detector using the request transition matrix and the detector using the request file distribution. The testbed for the experiments contains two major parts: the server machine and the client machine. The server machine has an emulated online store which follows the TPC-W specification for an E-commerce site under test. A client machine is a workload generator which contains emulated browsers (EBs) which can emulate end-user behaviors toward the emulated site. There are four types of faults and three types of special events injected into the data in order to evaluate the performance of these two detectors. From the transient response results, it is seen that both fault detectors can detect the injected faults, but only the one using the request transition matrix does not issue fault alarms for the special events. For performance evaluation and comparison, the detection time of the detector using the request transition matrix is less sensitive to workload changes and improve the detection time for two of the four types of faults. From the analysis of the ROC graphs for both detectors, the detector using the request transition matrix generally has better pairs of (TP, FP) and provides better performance because its ROC curves are closer to the top-left corner of the ROC graphs.

Chapter V presents the methods to construct a near real-time, online detection system using the request transition matrix and investigates the experimental results from the online testing. In order not to increase the overhead of the server machine,

an additional machine called monitoring site is added to the detection system and the detection program is executed on this machine. On the server side machine, the detection system requires execution of a program which can gets logged data from the web server. Due to using the pipe technique, results in getting the data from a web server with only minor configuration changes. No modification to the web server source code is required. Therefore, this makes deployment of the fault detector fairly easy. From the experimental results of the online testing, the online detector exhibits almost identical results to the ones from the off-line detector. It is able to not only distinguish between real faults and special events, but it also finds out which page has a problem while a fault occurs. This online detector exhibits all the four characteristics mentioned in the first chapter.

B. Conclusions

The current research intends to develop a user-visible fault detector for E-commerce sites. End-users change their behavior toward an E-commerce site if they encounter a user-visible fault and this behavioral change is reflected in the request transition matrix which is computed from data in server logs. Therefore, the proposed detector is basically detecting any abnormal change in the request transition matrix of web servers for use in fault detection, and it possesses all the following four characteristics: early detection, false alarm rate reduction, easy deployment, and fault localization. The following conclusions can be derived from the current research:

1. The proposed fault detection method is capable of distinguishing between real faults and special events and this reduces the overall fault alarm rate significantly. Also the detector can identify which specific page contains faults and this accomplishes the fault localization.

2. Compared with one of the most promising published user-visible fault detectors (the detector using χ^2-test to detect change of request file distribution) [38], the proposed user-visible fault detection system has comparable detection time for injected faults, For two of the four injected faults, the proposed system exhibits improved detection time.

3. If the require detection is less than about 30 minutes, the TP rate of the detector suffer while still maintaining low FP rate. The detector published in the literature exhibits marginally better, but still unacceptable TP rate, while exhibiting higher FP rate. For detection time of more than 30 minutes the conclusion from (1) and (2) above are valid.

4. The developed online fault detection system is easy to deploy requiring no major modification to the source code of a web server. Also the fault detection system does not increase the overhead of the monitored web server.

C. Limitations and Recommendations for Future Work

This research shows the feasibility of developing an User visible fault detector using the transition matrix. Some of the limitations of this work and the recommendations for future work are:

1. The developed fault detector can not handle very complicated user repeating actions, for example, an user repeated action which involves more than three requests.Future research can develop more sophisticated algorithms in order to handle such complicated repeated actions.

2. This research work lacks validation with real-world data which are difficult to obtain. Future research should obtain faulty data and data of some special

events from a real E-commerce sites in order to improve the algorithms and test the fault detector which suitable for real-world implementation.

3. This research work provides comparisons of two different user-visible fault detectors. There are still some other user-visible fault detectors, for example, Kiciman's "Pinpoint" fault detector [36]. Future research can develop methods to provide a thorough performance evaluation for all existing user-visible fault detectors.

4. As the detection time is reduced to below 30 minutes, the proposed detectors performance (TP) suffers significantly. No other detector appears to perform acceptable at the low detection times. Alternative approaches must be develop to handle such low detection times.

REFERENCES

[1] eMarketer, Inc. (2010, May). US retail E-commerce forecast: room to grow. Available: http://www.emarketer.com/Report.aspx?code=emarketer_2000672

[2] V. Sehgal, and Z. D. Wigder, Forrester Research. (2011, January). Forrester research online retail forecast, 2010 To 2015 (Western Europe). Available: http://www.forrester.com/rb/Research/research_online_retail_forecast,_2010_to_2015/q/id/58639/t/2

[3] H. Lee, CBC Marketing Research & Business Consulting Ltd,. (2009, December). Online-shopping market in China - adventurous kingdom for foreign SME. Avaiable: http://www.unifr.ch/intman/my_documents/NEWS/Lee%20-%20Online-Shopping%20in%20China.pdf

[4] Infonetics Research. (2006, December). The costs of downtime: North American medium businesses 2006. Available: http://ruthvictor.com/pdf/Network_Monitoring_NetGain/Whitepaper/Costs-of-Downtime-2006-Study.pdf

[5] R. Conrad, CNET. news.com. (2001, January). California power outages suspended–for now. Available: http://news.com.com/2100-1017-251167.html

[6] Meta Group. (2000, Oct.). IT performance engineering & measurement strategies: quantifying performance loss. Available: http://www.metagroup.com/cgi-bin/inetcgi/jsp/displayArticle.do?oid=18750

[7] The osCommerce community. (2010, November). osCommerce demo. Available: http://www.oscommerce.com/

[8] S. Pertet, and P. Narasimhan, "Causes of failure in web applications," *Carnegie Mellon University Parallel Data Lab Technical Report*, CMU-PDL-05-10, Dec. 2005.

[9] Tealeaf Technology, Inc. (2010, Decemeber). Social media buzz report: online holiday shopping. Available: http://portal.sliderocket.com/AHJJM/Tealeaf-Online-Shopping-Social-Media-Buzz-Report

[10] Equation Research, Inc., "When more web site visitors hurt your business: are you ready for peak traffic?" *Compuware Company 2010 White Papers*, May 2010.

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

[12] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861-874, June 2006.

[13] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*, 1st edition, Hoboken, NJ: John Wiley & Sons, Inc., 2004.

[14] D. A. Menasce ,and V. A. F. Almeida, *Capacity Planning for Web Performance*, 1st edition, Upper Saddle River, NJ: Prentice-Hall Inc., 1998.

[15] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing model based network server performance control," in *Proc. 23rd IEEE Symposium of Real-Time Systems*, Austin, Feb. 2003, pp. 81-90.

[16] Y. Lu, T. Abdelzaher, and C. Lu, "Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers," in *Proc. 9th IEEE sym-*

posium of Real-Time and Embedded Technology and Applications, Toronto, May 2003, pp. 208-217.

[17] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. Hellerstein, and S. Parekh, "Online response time optimization of Apache web server," in *Proc. 11th International Conference on Quality of Service* , Berkeley, 2003, pp. 461-478.

[18] N. Gandhi, D. M. Tilbury, Y. Diao, J. Hellerstein, and S. Parekh, "MIMO control of an Apache web server: modeling and controller design," in *Proc. American Control Conference*, vol. 6, pp. 4922-4927, Nov. 2002.

[19] P. Pradhan, R. Tewari, S. Sahu, A. Chandra, and P. Shenoy, "An observation based approach towards selfmanaging web servers," in *Tenth IEEE International Workshop Quality of Service*, Miami Beach, August 2002, pp. 13-22.

[20] B. Schroeder, and M. Harchol-Balter, "Web servers under overload: how scheduling can help," *ACM Transactions on Internet Technology (TOIT)*, vol. 6, no. 1, pp. 20-52, Feb. 2006.

[21] H. Chen, and P. Mohapatra, "Session-based overload control in QoS-aware web servers," in *Proc. of INFOCOM, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 516-524, Nov. 2002.

[22] F. Sultan, A. Bohra, P. Gallard, I. Neamtiu, S. Smaldone, Y. Pan. Neamtiu, and L. Iftode, "Nonintrusive failure detection and recovery for Internet services using backdoors," *Rutgers University Technical Report*, DCS-TR-524, Dec. 2003.

[23] R. Klemm, and N. Singh, "Automatic failure detection, logging, and recovery for high-availability Java servers," in *13th International Symposium of Software Reliability Engineering, ISSRE 2003*, Washington DC, Jan. 2003, pp. 79-90.

[24] X. Li, R. R. Martin, K. Nagaraja, T. D. Nguyen, and B. Zhang, "Mendosus: a SAN-based fault-injection test-bed for construction of highly available network services," in *Proc. 1st Workshop on Novel Uses of System Area Networks (SAN-1)*, Cambridge, Feb. 2002, pp. 70-79.

[25] K. Nagaraja, X. Li, B. Zhang, R. Bianchini, R. P. Martin, T. D. Nguyen, "Using fault injection and modeling to evaluate the performability of cluster-based services," in *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, March 2003, pp. 17-31.

[26] C. Fu, R. P. Martin, K. Nagaraja, T. D. Nguyen, B. G. Ryder, and D. G. Wonnacott, "Compiler directed program-fault coverage for highly available Java Internet services," in *Proc. International Conference on Dependable Systems and Networks (DSN, IPDS track)*, San Francisco, June 2003, pp. 595 - 604 .

[27] C. Fu, B. Ryder, A. Milanova, and D. Wonnacott, "Testing of Java web services for robustness," in *Proc. of the International Symposium on Software Testing and Analysis (ISSTA 2004)*, Boston, July, 2004, pp. 23-34.

[28] A. Ward, P. Glynn, and K. Richardson, "Internet service performance failure detection," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 38-43, Dec. 1998.

[29] S. Garg, K. Vaidyanathan, A. V. Moorsel, and K. S. Trivedi , "A methodology for detection and estimation of software aging," in *Proc. 9th Intl. Symposium on Software Reliability Engineering*, Paderborn, Nov. 1998, pp. 282-294.

[30] K. S. Trivedi, K. Vaidyanathan, and K. Goseva-Popstojanova, "Modeling and analysis of software aging and rejuvenation," in *33rd Ann. Simulation Symposium, 2000. (SS 2000)*, Washington DC, Apr. 2000, pp. 270-279.

[31] K. C. Gross, V. Bhardwaj, and R. Bickford, "Proactive detection of software aging mechanisms in performance critical computers," in *IEEE 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, Greenbelt, Dec. 2003, pp. 17-23.

[32] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating instrumentation data to system states: A buliding block for automated diagnosis and control," in *6th USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, Dec. 2004, pp. 231-244.

[33] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, "Software aging and multifractality of memory resources," in *IEEE 2003 International Conference on Dependable Systems and Networks (DSN'03)*, San Francisco, June 2003, pp. 721 - 730.

[34] S. Sprenkle, E. Gibson, S. Sampath, L. Pollock, "Automated replay and failure detection for web applications," in *Proc. 20th IEEE/ACM International Conference on Automated Software Engineering*, Long Beach, Nov. 2005, pp. 253-262.

[35] M. Y. Chen, E. Kiciman, E. Fratkin, E. Brewer and A. Fox, "Pinpoint: problem determination in large, dynamic, Internet services," in *Proc. 2002 International Conference on Dependable Systems and Networks (IPDS Track)*, Washington DC, Dec. 2002, pp. 595-604.

[36] E. Kiciman, and A. Fox, "Detecting application-level failures in component-based Internet services," *IEEE Transactions on Neural Networks: Special Issue on Adaptive Learning Systems in Communication Networks*, vol. 16, no. 5, pp. 1027-1041, September 2005.

[37] G. Candea, E. Kiciman, S. Zhang, P. Keyani, A. Fox, "JAGR: an autonomous self-recovering application server," in *Proc. 5th International Workshop on Active Middleware Service*, Seattle, June 2003, pp. 168-178.

[38] B. C. Ling, E. Kiciman, A. Fox, "Session state: beyond soft state," in *Proc. 1st Symposium on Networked Systems Design and Implementation*, San Francisco, March 2004, pp. 295-308.

[39] P. Bodik, G. Friedman, L. Biewald, H. Levine, G. Candea, K. Patel, G. Tolle, J. Hui, A. Fox, M. I. Jordan, and D. Patterson, "Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization," in *Proc. Second International Conference on Autonomic Computing, 2005 (ICAC 2005)*, Seattle, June 2005, pp. 89-100.

[40] Mindcraft, Inc. (1995, March). WebStone. Available: http://mindcraft.com/webstone

[41] B. Paul, and C. Mark,"Generating representative web workloads for network and server performance evaluation," *ACM SIGMETRICS*, vol.26, no.1, pp. 151-160, June 1998.

[42] D. Menasce,"TPC-W, a benchmark for E-commerce," *IEEE Internet Computing*, vol. 6, no. 3, pp. 83-87, May/June 2002.

[43] Standard Performance Evaluation Corp. (2010, January). SPECweb2005. Available: http://www.spec.org/web2005/

[44] J. J. Gertler, *Fault Detection and Diagnosis in Engineering Systems*, New York, NY: Marcel Dekker, 1998.

[45] The Transaction Processing Performance Council. (2002, Feburary). TPC-W V1.8. Available: http://www.tpc.org/

[46] H. W. Cain, and R. Rajwar, "An architectural evaluation of Java TPC-W," in *Proc. Seventh International Symposium on High-Performance Computer Architecture*, Monterrey, January 2001, pp. 229-240.

[47] The Apache Software Fundation. (2010, October). Apache HTTP server. Available: http://httpd.apache.org/

[48] The Apache Software Fundation. (2005, March). Apache Tomcat. Avaiable: http://tomcat.apache.org/

[49] The Oracle Inc. (2005, March). Oracle JDK. Available: http://www.oracle.com/technetwork/java/javase/overview/index.html

[50] MySQL Corp. (2009, December). MySQL database server. Available: http://www.mysql.com/

[51] The Apache Software Fundation. (2005, April). The Apache Tomcat connector. Available: http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html

[52] MySQL Corp. (2005, March). MySQL Connector/J. Available: http://www.mysql.com/downloads/connector/j/

[53] Easy Store Hosting Inc. (2010, December). *easystorehosting* Demo. Avaiable: http://www.easystorehosting.com/

[54] D. Menascé , V. Almeida , R. Fonseca , F. Ribeiro, R. Riedi, and W. Meira Jr., "In search of invariants for E-business workloads," in *Proc. 2nd ACM Conference on Electronic Commerce*,Minneapolis, 2000, pp. 56-65.

[55] M. Mitchell, J. Oldham, and A. Samuel, *Advanced Linux Programming*, Indianapolis, IN: New Riders Pub, 2001.

[56] Apache Software Fundation. (2011, May). Apache module*mod_log_config*," Available: http://httpd.apache.org/docs/2.0/mod/mod_log_config.html

[57] Apache Software Fundation. (2011, May) *rotatelogs* - piped logging program to rotate apache logs. Available: http://httpd.apache.org/docs/2.0/programs/rotatelogs.html

VITA

Pang-Chun Chu received his Bachelor of Science (BS) Degree in Mechanical Engineering from National Chung Hsing University, Taiwan in 1996 and obtained his Master of Science (MS) degree in Mechanical Engineering form National Chiao Tung University, Taiwan in 1998. He worked as an Engineer for about one year in Hon-Hai Precision Industry Inc., Taiwan from 2000 - 2001. He joined Texas A&M University, College Station, TX, USA and received his Ph.D in Mechanical Engineering in December 2011.

The typist for this thesis was Pang-Chun Chu.