An Integrated Framework for Optimizing Automatic Monitoring Systems in Large IT Infrastructures

Liang Tang Tao Li
School of Computer Science
Florida International University
11200 S.W. 8th Street
Miami, Florida, 33199
U.S.A
{ltang002,taoli}@cs.fiu.edu

Larisa Shwartz, Florian Pinel
IBM Watson Research Center
1101 Kitchawan Rd
Yorktown Heights, NY, 10598
U.S.A
{lshwart,pinel}@us.ibm.com

Genady Ya. Grabarnik,
Dept. Math & Computer Science
St. John's University
Queens, NY
U.S.A
grabarng@stjohns.edu

ABSTRACT

The competitive business climate and the complexity of IT environments dictate efficient and cost-effective service delivery and support of IT services. These are largely achieved by automating routine maintenance procedures, including problem detection, determination and resolution. System monitoring provides an effective and reliable means for problem detection. Coupled with automated ticket creation, it ensures that a degradation of the vital signs, defined by acceptable thresholds or monitoring conditions, is flagged as a problem candidate and sent to supporting personnel as an incident ticket. This paper describes an integrated framework for minimizing false positive tickets and maximizing the monitoring coverage for system faults.

In particular, the integrated framework defines monitoring conditions and the optimal corresponding delay times based on an off-line analysis of historical alerts and incident tickets. Potential monitoring conditions are built on a set of predictive rules which are automatically generated by a rule-based learning algorithm with coverage, confidence and rule complexity criteria. These conditions and delay times are propagated as configurations into run-time monitoring systems. Moreover, a part of misconfigured monitoring conditions can be corrected according to false negative tickets that are discovered by another text classification algorithm in this framework. This paper also provides implementation details of a program product that uses this framework and shows some illustrative examples of successful results.

Categories and Subject Descriptors

H.2.8 [**Database applications**]: Data mining

Keywords

Ticket analysis, System monitoring

KDD'13, August 11–14, 2013, Chicago, Illinois, USA.

1. INTRODUCTION

IT service providers are facing an increasingly intense competitive landscape and growing industry requirements. In their quest to maximize customer satisfaction, Service Providers seek to employ business intelligence solutions, which provide deep analysis, orchestration of business processes and capabilities for optimizing the level of service and cost. IT Infrastructure Library (ITIL) addresses monitoring as a continual cycle of monitoring, reporting and subsequent action that provides measurement and control of services [4].

Modern forms of distributed computing (e.g., cloud) provided some standardization of the initial configuration of the hardware and software. In order to enable most enterprise level applications, however, an individual infrastructure for the given application must be created and maintained on behalf of each outsourcing customer. This requirement creates great variability in the services provided by IT support teams. The aforementioned issues largely contribute to the fact that routine maintenance of the information systems remains semi-automated and manually performed. System monitoring is an automated reactive system that provides an effective and reliable means of ensuring that degradation of the vital signs, defined by acceptable thresholds or monitoring conditions (situations), is flagged as a problem candidate (monitoring event) and sent to the service delivery teams as an incident ticket.

Defining monitoring conditions (situations) requires the knowledge of a particular system and its relationships with other hardware and software systems. It is a known practice to define conservative conditions in nature, thus erring on the side of caution. This practice leads to a large number of tickets that require no action (false positives). Continuous updating of modern IT infrastructures also leads to a number of system faults that are not captured by system monitoring (false negatives). To minimize the number of false positives and false negatives, this paper presents an integrated framework for optimizing automatic monitoring systems in large and dynamic IT infrastructures. It utilizes learning based approaches on historical monitoring events and incident tickets to help system administrators improve monitoring condition definitions. This framework is implemented as an *event and ticket analysis system*, which becomes part of the IBM IT service management platform. The system is deployed and maintained at several IBM service centers. The end-users are the system administrators who are working with IBM Tivoli monitoring [2]. The ana-

lyzed results of the system have also been deployed at several IBM customer accounts. A customer account stands for an enterprise IT infrastructure which is often constructed by over one thousand machines. The monitoring quality was improved significantly after deployment for several months.

1.1 Contributions

This paper describes an integrated framework for optimizing automatic monitoring systems in large and dynamic IT infrastructures. The innovations implemented through this framework include:

- Investigation of the false positive alerts and false negative alerts of automatic monitoring in modern real IT infrastructures.
- Development of an integrated system for minimizing the number of false positives and false negatives based on two learning-based approaches with historical monitoring events and incident tickets.
- Deployment of experiments on real large IT environments and historical data. The experimental results demonstrated the benefits of our improving the qualify of the IT infrastructure monitoring.

The rest of the paper is organized as follows: Section 2 provides a description of the problem settings and introduces the main notations used in the paper. Section 3 presents our developed framework and system for minimizing false positive and false negative alerts. In Section 4, we present our empirical studies on IBM Tivoli monitoring systems with real IT incident tickets. Section 5 summarizes the related work for text classification and system event mining for system management. sSection 6 offers our paper's conclusion.

2. BACKGROUND

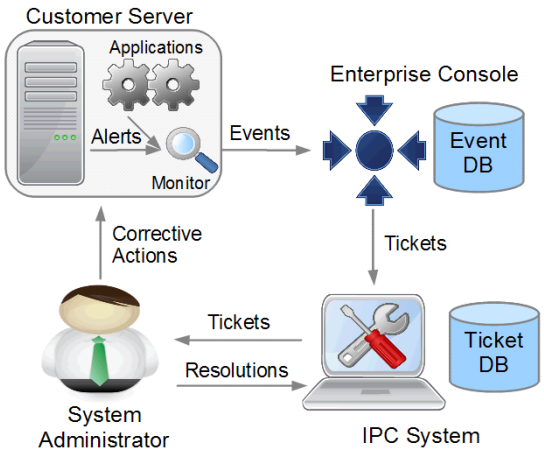


Figure 1: Problem Detection, Determination and Resolution

The typical workflow of problem detection, determination and resolution for the IT service provider is prescribed by the ITIL specification [4], and illustrated in Figure 1. Detection is usually provided by monitoring software running on the servers of an account, which computes metrics for the hardware and software performance at regular intervals. The metrics are then compared to acceptable thresholds, known as *monitoring situations*, and any violation results in an alert. If the alert persists beyond a certain delay specified in the situation, the monitor emits an event. Events coming from an account's entire IT environment are consolidated in an enterprise console. The console uses rule-, case- or knowledge-based engines to analyze the monitoring events and decide whether to open a service ticket in the Incident, Problem, Change (IPC) system. Additional tickets are created upon customer request. The information accumulated in the ticket is used by the System Administrators (SAs) for problem determination and resolution. As part of the service contracts between the customer and the service provider, the SLA (Service Level Agreement) specifies the maximum resolution times for various categories of tickets.

Performing a detailed analysis of IT system usage is time-consuming, so SAs often rely on default monitoring situations. Furthermore, IT system usage is likely to change over time. This often results in a large number of alerts and tickets, which can be categorized in Table 1.

Table 1: Definitions for Alert, Event and Ticket

False Positive Alert	An alert for which the system administrator does not need to take any action.
False Negative Alert	A missed alert that is not captured due to inappropriate monitoring configuration.
False Alert	False positive alert
Real Alert	An alert that requires the system administrator to fix the corresponding problem on the server.
Alert Duration	The length of time from an alert creation to its clearing.
Transient Alert	An alert that is automatically cleared before the technician opens its corresponding ticket.
Event	The notification of an alert to the Enterprise Console.
False Positive Ticket	A ticket created from a false positive alert.
False Negative Ticket	A ticket created manually identifying a condition that should have been capture by automatic monitoring.
False Ticket	A ticket created from a false alert.
Real Ticket	A ticket created from a real alert.

Whether a ticket is real or false is determined by the resolution message entered in the ticket tracking database by the system administrator it was assigned to. It is not rare to observe entire categories of alerts, such as CPU or paging utilization alerts, that are almost exclusively false positives. When reading the resolution messages one by one, it can be simple to find an explanation: Anti-virus processes cause prolonged CPU spikes at regular intervals; databases may reserve large amount of disk space in advance, making the monitors believe the system is running out of storage. With only slightly more effort, one can also fine-tune the thresholds of certain numerical monitored metrics, such as the metrics involved in paging utilization measurement. There are rarely enough human resources, however, to correct the monitoring situations one system at a time, and we need an algorithm capable of discovering these usage-specific rules. There has been a great deal of effort spent on developing the monitoring conditions (situations) that can identify potentially unsafe functioning of the system [14] [24]. It is understandably difficult, however, to recognize and quantify influential factors in the malfunctioning of a complex system. Therefore classical monitoring tends to rely on periodical probing of a system for conditions that could potentially contribute to the system's misbehavior. Upon detection of the predefined conditions, the monitoring systems trig-

ger events that automatically generate incident tickets. In this paper, we study the problem of improving the quality of monitoring systems based on the analysis for historical monitoring events and incident tickets.

3. EVENT AND TICKET ANALYSIS

This section presents the integrated framework of our developed system. We first briefly describe the system overview and then discuss its two main functionalities.

3.1 Overview

Figure 2 shows an overview architecture for the integrated framework. The framework is implemented as an event and ticket analysis system (called *Event & Ticket Analysis Portal* in this figure), which becomes part of the IBM IT service management platform. This system connects to multiple ticketing systems and multiple event data warehouses. The ticketing system stores and manages incident tickets. Each IBM IT service's enterprise customer has its own ticketing system, which is the data source of the historical tickets. `NetCool` is the event data warehouse for storing and analyzing the monitoring events at IBM IT service centers. A *NetCool* manages several enterprise IT infrastructures, which is the data source of the historical events. Our system integrates the ticket data and the event data from the two systems to generate the analysis reports. Figure 3 is

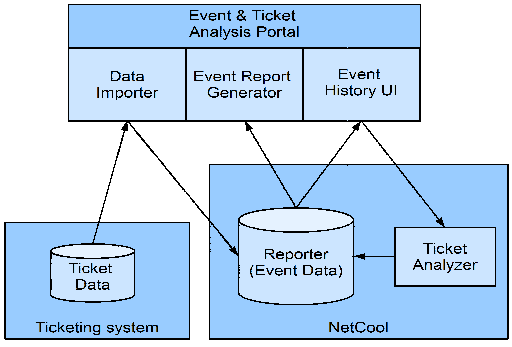


Figure 2: the Architecture Overview

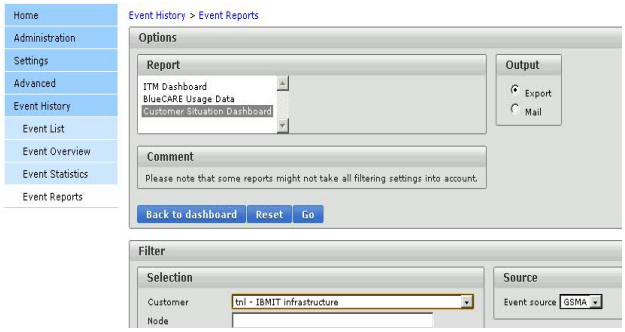


Figure 3: the Web Interface

a screenshot of the web user interface. By selecting different data sources, the system generates analysis reports for different customer accounts. A customer account stands for an enterprise IT infrastructure. Most enterprise customers of IBM have over one thousand machines in their infrastructures. The generated reports help the system administrators define better monitoring situations to improve the quality of system monitoring. The reports are delivered as spreadsheets that can be downloaded or emailed to the end-users (see Figure 3). Usually, for every customer account, the system administrators import recent data and generate the reports once a month. The configurations of the automatic monitoring system are periodically updated based on the generated reports. False positive alerts and false negative alerts are two important components in the reports.

3.2 False Positive Alerts

Our goal is to eliminate as many false alerts as possible while retaining all real alerts. A naive solution is to build a predictive classifier and adjust the monitoring situations according to the classifier. Unfortunately, no prediction approach can guarantee 100% success for real alerts, but a single missed one may cause serious problems, such as system crashes or data loss.

The vast majority of the false positive alerts are transient, such as temporary spikes in CPU and paging utilization, service restarts, and server reboots. These transient alerts automatically disappear after a while, but their tickets are created in the ticketing system. When system administrators open the tickets and log on the server, they cannot find the problem described by those tickets. Figure 4 shows the duration histogram of false positive alerts raised by one monitoring situation. This particular situation checks the status of a service and generates an alert without delay if the service is stopped or shutdown. These false positive alerts are collected from one server of a customer account for 3 months. As shown by this figure, more than 75% of the alerts can be cleared automatically by waiting 20 minutes. It is possible

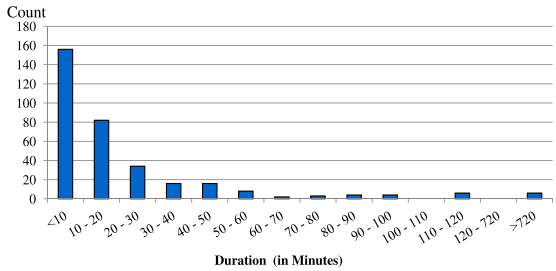


Figure 4: False Positive Alert Duration

for a transient alert to be caused by a real system problem. From the perspective of the system administrators, however, if the problem cannot be found when logging on the server, there is nothing they can do with the alert, no matter what happened before. Some transient alerts may be indications of future real alerts and may be useful. But if those real alerts arise later on, the monitoring system will detect them even if the transient alerts were ignored. Therefore, all transient alerts are considered false negative.

Eliminating False Positive Alerts Safely

Our solution first predicts whether an alert is real or false. If it is predicted as real, a ticket will be created. Otherwise, the ticket creation will be postponed. Our solution also determines how long is it to be postponed. Even if a real alert is incorrectly classified as false, its ticket will eventually be created before violating the SLA. Figure 5 shows a flowchart of an incoming event. It reveals two key problems for this approach: (1) How to predict whether an alert is false or

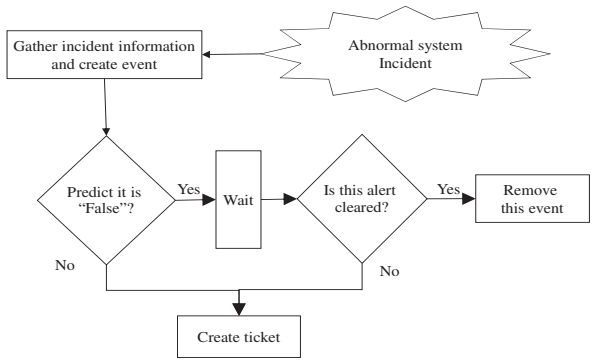


Figure 5: Flowchart for Ticket Creation

real? (2) If an alert is identified as false, what waiting time should be applied before ticket creation?

In our approach, the predictor is implemented by a rule-based classifier based on the historical tickets and events. The ground truth of the events is obtained from the associated tickets. Each historical ticket has one column that suggests this ticket is real or false. This column is manually filled by the system administrators and stored in the ticketing system. There are two reasons for choosing a rule-based predictor. First, each monitoring situation is equivalent to a quantitative rule. The predictor can be directly implemented in the existing monitoring system. Other sophisticated classification algorithms, such as *support vector machine* and *neural network*, may have a higher precision in predicting. Their classifiers, however, are very difficult to implement as monitoring situations in real systems. Second, a rule-based predictor is easily verifiable by the end users. Other complicated classification models represented by linear/non-linear equations or neural networks are very hard for end users to verify. If the analyzed results could not be verified by the system administrators, they would not be utilized in real production servers.

Predictive Rule

The alert predictor roughly assigns a label to each alert, "false" or "real." It is built on a set of predictive rules that are automatically generated by a rule-based learning algorithm [27] based on historical events and alert tickets. Example 1 shows a predictive rule, where "*PROC_CPU_TIME*" is the CPU usage of a process. Here "*PROC_NAME*" is the name of the process.

EXAMPLE 1. **if** PROC_CPU_TIME > 50% **and** PROC_NAME = 'Rtvscan', **then** this alert is false.

A predictive rule consists of a rule condition and an alert label. A rule condition is a conjunction of *literals*, where each *literal* is composed of an event attribute, a relational operator and a constant value. In Example 1, "*PROC_CPU_TIME > 50%*" and "*PROC_NAME = 'Rtvscan'*" are two *literals*, where "*PROC_CPU_TIME*" and "*PROC_NAME*" are event attributes, ">" and "=" are relational operators, and "50%" and "*Rtvscan*" are constant values. If an alert event satisfies a rule condition, we call this alert covered by this rule. Since we only need predictive rules for false alerts, the alert label in our case is always "false."

Predictive Rule Generation

The rule-based learning algorithm [27] first creates all *literals* by scanning historical events. Then, it applies a breadth-first search for enumerating all *literals* in finding predictive rules, i.e., those rules having predictive power. This algorithm has two criteria to quantify the minimum predictive power: the minimum confidence $minconf$ and the minimum support $minsup$ [27]. In our case, $minconf$ is the minimum ratio of the numbers of the covered false alerts and all alerts covered by the rule, and $minsup$ is the minimum ratio of the number of alerts covered by the rule and the total number of alerts. The two criteria govern the performance of our method, defined as the total number of removed false alerts. To achieve the best performance, we loop through the values of $minconf$ and $minsup$ and compute their performances.

Predictive Rule Selection

Although the predictive rule learning algorithm can learn many rules from the historical events with tickets, we only select those with strong predictive power. In our solution, Laplace accuracy [36] [20] [17] is used for estimating the predictive power of a rule. According to the SLA, real tickets must be acknowledged and resolved within a certain time. The maximum allowed delay time is specified by a user-oriented parameter $delay_{max}$ for each rule. In the calculation of Laplace accuracy, those false alerts are treated as real alerts if their durations are greater than $delay_{max}$. $delay_{max}$ is given by the system administrators according to the severity of system incidents and the SLA.

Another issue is rule redundancy. For example, let us consider the two predictive rules:

X. *PROC_CPU_TIME* > 50% ***and*** *PROC_NAME* = 'Rtvscan'
Y. *PROC_CPU_TIME* > 60% ***and*** *PROC_NAME* = 'Rtvscan'

Clearly, if an alert satisfies Rule Y, then it must satisfy Rule X as well. In other words, Rule Y is more specific than Rule X. If Rule Y has a lower accuracy than Rule X, then Rule Y is redundant given Rule X (but Rule X is not redundant given Rule Y). In our solution, we perform redundant rule pruning to discard the more specific rules with lower accuracies. The detailed algorithm is described in [31].

Calculating Waiting Time for Each Rule

Waiting time is the duration by which tickets should be postponed if their corresponding alerts are classified as false. It is not unique for all monitoring situations. Since an alert can be covered by multiple predictive rules, we set up different waiting times for each of them. The waiting time can be transformed into two parameters in monitoring systems, the length of the polling interval with the minimum polling count [3]. For example, the situation described in Example 1 predicts false alerts about CPU utilization of 'Rtvscan.' We can also find another predictive rule as follows:

if *PROC_CPU_TIME > 50%* ***and*** *PROC_NAME = 'perl logqueue.pl',* ***then*** *this alert is false.*

The job of 'perl', however, is different from that of 'Rtvscan.' Their durations are not the same, and the waiting time will differ accordingly. In order to remove as many false alerts as possible , we set the waiting time of a selected rule as the

longest duration of the transient alerts covered by it. For a selected predictive rule p, its waiting time is

$$wait_p = \max_{e \in \mathcal{F}_p} e.duration,$$

where $\mathcal{F}_p = \{e | e \in \mathcal{F}, isCovered(p, e) =' true'\}$, and $\mathcal{F}$ is the set of transient events. Clearly, for any rule $p \in \mathcal{P}$, $wait_p$ has a upper bound, $wait_p \leq delay_{max}$. Therefore, no ticket can be postponed for more than $delay_{max}$.

3.3 False Negative Alerts

False negative alerts are the missing alerts that are not captured by the monitoring system due to some misconfiguration. Real-world IT infrastructures are often over-monitored. False negative alerts are much fewer than false positive alerts. Since the number of false negative alerts is quite small, we only focus on the methodologies for discovering them with their corresponding monitoring situations. The system administrators can easily correct the misconfiguration by referring the results. The false negative tickets are recorded by the system administrators in the manual tickets. Each manual ticket consists of several textual messages that describe the detailed problem. In addition to system fault issues, manual tickets also track many other customer requests such as asking for resetting database passwords, installing a new web server and so on. The customer request is the majority of the manual tickets. In our system the work for false negative alerts is to find out those monitoring related tickets among all manual tickets. This problem is formed as a binary text classification problem. Given an incident ticket, our method classifies it into "1" or "0", where "1" indicates this ticket is a false negative ticket, otherwise it is not. For each monitoring situation, we build a binary text classifier.

There are two challenges for building the classification model. First, the manual ticket data is highly imbalanced since most of the manual tickets are customer requests and only very few are false negative tickets. Figure 6 shows various system situation issues in two manual ticket sets. This manual ticket set is collected from a large customer account in IBM IT service centers. The first month has 9854 manual tickets and the second month has 10109 manual tickets overall. As shown in this figure, only about 1% manual tickets are false negatives. Second, labeled data is very limited.

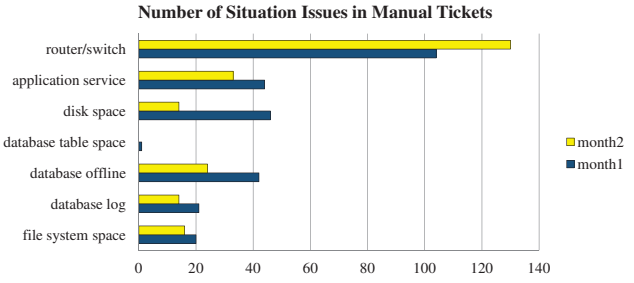


Figure 6: Number of Situation Tickets

Most system administrators are only working on some parts of incident tickets. Only a few experts can label all tickets.

Selective Ticket Labeling

It is time-consuming for human experts to scan all manual tickets and label their classes for training. In our approach, we only select a small proportion of tickets for labeling. A naive method is randomly selecting a subset of the manual tickets as the training data. However, the selection is crucial to the highly imbalanced data. Since the monitoring related tickets are very rare, the randomly selected training data would probably not contain any monitoring related ticket. As a result, the classification model cannot be trained well. On the other hand, we do not know which ticket is related to monitoring or not before we obtain the tickets' class labels. To solve this problem, we utilize domain words in system management for the training ticket selection. The domain words are some proper nouns or verbs that indicate the scope of the system issues. For example, everyone uses "DB2" to indicate the concept of IBM DB2 database. If a ticket is about the DB2 issue, it must contain the word "DB2". "DB2" is a domain word. There are not many variabilities for the concepts described by the domain words. Therefore, those domain words is helpful to reduce the ticket candidates for labeling. Table 2 lists examples of the domain words with their corresponding situations. The domain words can be obtained from the experts or related documents.

Table 2: Domain Word Examples

Situation Issue	Words
DB2 tablespace Utilization	DB2, tablespace
File System Space Utilization	space,file
Disk Space Capacity	space,drive
Service Not Available	service,down
Router/Switch Down	router

In the training ticket selection, we first compute the relevance score of each manual ticket and ranks all the tickets based on the score, and then select the top k tickets in the ranked list, where k is a predefined parameter. Given a ticket T, the relevance score is computed as follows:

$$score(T) = \max\{|w(T) \cap M_1|, ..., |w(T) \cap M_l|\},$$

where $w(T)$ is the word set of ticket T, l is the number of predefined situations, M_i is the given domain word set for the i-th situation, $i = 1, ..., l$. Intuitively, the score is the largest number of the common words between the ticket and the domain words.

In the evaluation section of this paper, we consider a baseline method that only uses the domain words to identify situation tickets by simple word-matching. In dual supervision learning[26], the domain words are seen as the labeled features , which can also be used in active learning for selecting unlabeled data instances. But in our application, we have only the positive features but no negative features, and the data is highly imbalanced. Therefore, the uncertainty-based approach and the density-based approach in active learning are not appropriate for our system.

Classification Model Building

The situation ticket is identified by applying a SVM classification model [28] on the ticket texts. For training this model, we have two types of input data: 1) the selectively labeled tickets, and 2) the domain words. To utilize the domain words, we treat each domain word as a *pseudo-ticket* and put all *pseudo-tickets* into the training ticket set. To deal with the imbalanced data, the minority class tickets are over-sampled until the number of positive tickets is equal to the number of the negative tickets [9]. Figure 7 shows the flow chart for building the SVM classification model.

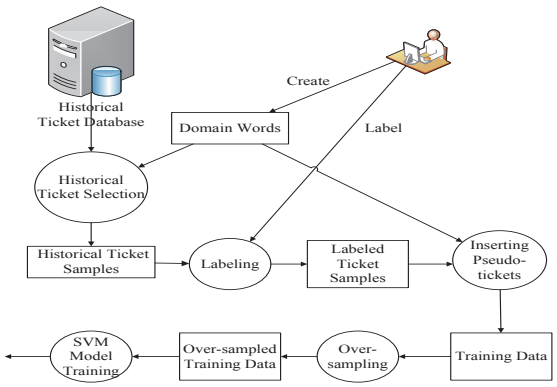


Figure 7: Flow Chart of Classification Model

4. EVALUATION

This section presents empirical studies for our system. The system and the analysis results have been deployed for several customer accounts of IBM IT service. The empirical studies have two types of evaluation. The first type of evaluation is on the collected historical data to validate the performance of the algorithms. The second on is on the production severs of IBM customers to validate the effectiveness on real IT infrastructures.

4.1 Evaluation on Historical Data

Our system is developed by Java 1.6. This testing machine is Windows XP with Intel Core 2 Duo CPU 2.4GHz and 3GB of RAM. Experimental monitoring events and tickets are collected from production servers of the IBM Tivoli Monitoring system [2], summarized in Table 3. The data set of each account covers a period of 3 months. $|\mathcal{D}|$ is the number of events that generated tickets in the ticketing systems. N_{non} is the number of false events in all ticketed events. # Attributes is the total number of attributes of all events. # Situations is the number of monitoring situations. # Nodes is the number of monitored servers. In addition to the auto-generated tickets, we also collect manual tickets from two months. The first month has 9584 manual tickets. The second month has 10109 manual tickets.

Table 3: Data Summary

Data Set	$\|\mathcal{D}\|$	N_{non}	# Attributes	# Situations	# Nodes
Account1	50,377	39,971	1082	320	1212

4.1.1 False Positive Alerts

Performance Measure

There are two performance measures:

- FP: The number of false tickets eliminated.
- FD: The number of real tickets postponed.

To achieve a better performance, a system should have a larger FP with a smaller FD. We split each data set into the training part and the testing part. "Testing Data Ratio" is the fraction of the testing part in the data set, and the rest is the training part. For example, "Testing Data Ratio=0.9" means that 90% of the data is used for testing and 10% is used for training. All FP and FD are only evaluated for the testing part.

Overall Performance

Based on the experience of the system administrators, we set $delay_{max} = 360$ minutes for all monitoring situations. Figure 8 presents the experimental results. Our method eliminates more than 75% of the false alerts and only postpones less than 3% of the real tickets.

Comparing with *Revalidate*

Since most alert detection methods cannot guarantee no false negatives, we only compare our method with the idea mentioned in [8], *Revalidate*, which revalidates the status of events and postpones all tickets. *Revalidate* has only one parameter, the postponement time, which is the maximum allowed delay time $delay_{max}$. Figure 8c compares the respective performance of our method and *Revalidate*, where each point corresponds to a different test data ratio. While *Revalidate* is clearly better in terms of elimination of false alerts, it postpones all real tickets, the postponement volume being 1000 to 10000 times larger than our method.

Predictive Rules

Tables 4 lists several discovered predictive rules for false alerts, where $wait_p$ is the delay time for a rule, FP_p is the number of false alerts eliminated by a rule in the testing data, and FD_p is the number of real tickets postponed by a rule in the testing data.

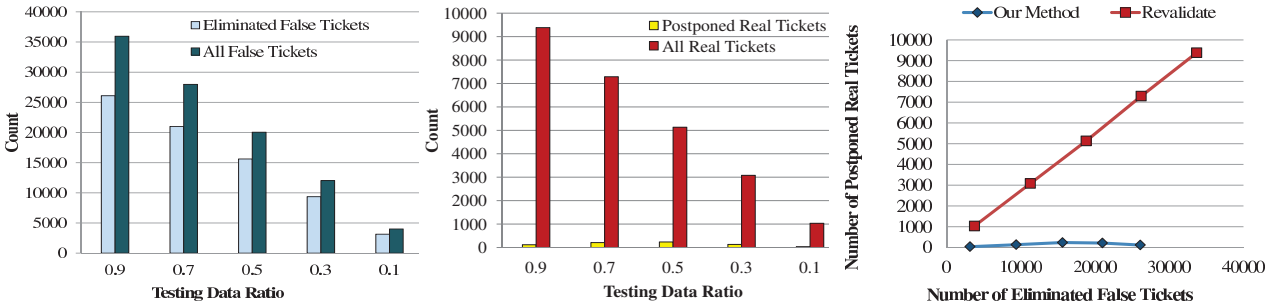
4.1.2 False Negative Alerts

The effectiveness is evaluated by the accuracy of the situation discovery. The accuracy is measured by precision, recall and F1score, which are the standard accuracy metrics in classification problems [25]. We use one month's tickets as the training data, and the other month's tickets as the testing data. We first test the accuracy of the word-match method. The words are predefined in Table 6.

Figure 9 shows the tested F1 scores [28] of four monitoring situations about file system space issue, disk space issue, service availability and router/swith issues. Our method is denoted as "Selective", the second baseline method is denoted as "Random". The "Random" method randomly selects a subset of manual tickets as the training data for building the SVM model. The domain words for our "Selective" method are shown in Table 6. As shown by those figures, the "Random" method can only achieve the same accuracy of our method when the number of training tickets is large (above 5000). This is because the real situation tickets are in the minority of the training data set. The training tickets in "Random" cannot capture real situation tickets unless the training data is large. If the training data is large, however, labeling would be time-consuming for humans.

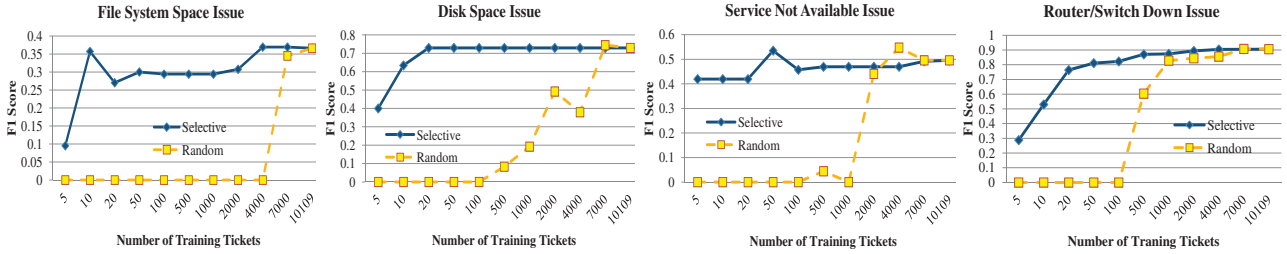
4.2 Evaluation on Production Servers

Our developed system and analytic results have been deployed into several customer accounts of IBM IT services. We track the changes on those customer accounts after the deployment. Figure 10 shows the deployed results of one account in three months. Account1 is the account that provides the historical data in the previous evaluation of this paper. This customer account is a large financial company in the United States. Its production servers are used mainly to support financial investments. The deployment of our work is a step-by-step approach. In the first month, the deployment is only on a small group of testing and devel-



(a) Eliminated False Positive Tickets (b) Postponed Real Tickets (c) Comparison with *Revalidate*

Figure 8: Eliminating False Positive Tickets



(a) File system Space (b) Disk space (c) Service not available (d) Router/switch down

Figure 9: Accuracy for Situation Discovery

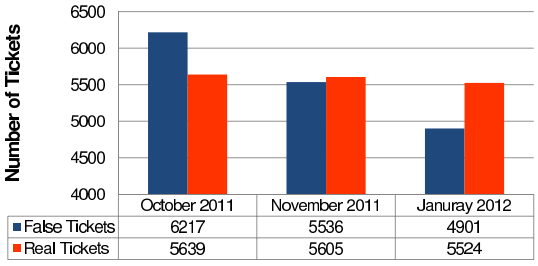


Figure 10: Ticket Volume Changes on Account1

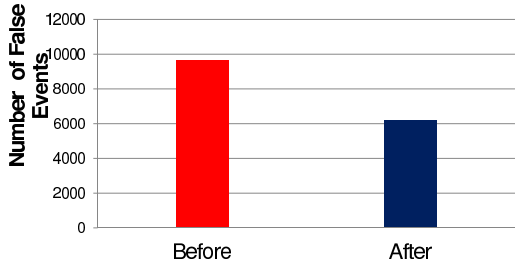


Figure 11: Event Volume Changes on Account2

opment machines. Then it spreads to a wide area of its IT environment. Hence, in Figure 10, the effect of our work gradually appears in three months. Although this company's IT infrastructure changes every day, compared to the changes of real tickets, the reduction for the false tickets is still obvious. The total number of false positive tickets has been reduced by 21%.

Figure 11 shows the evaluation results for another customer account of IBM IT services. They compare the number of false alerts before deployment and after deployment. Before the deployment, this account has many inappropriate CPU and networking monitoring situations, which produce a large number of false alerts every day. By adjusting those monitoring situations according to our analysis reports, more than 30% of the false alerts are eliminated.

Table 5 shows a sample list of discovered false negative tickets with their monitoring situations on Account1. For privacy issues, the administrators' names and the server names are replaced by "xxx". Most of the false negative tickets are caused by some new servers or new databases that are not added into the configuration of monitoring systems. When the new servers and new databases incur system faults or issues, only the database administrators or storage administrators discover them and create the manual tickets. The false negative tickets are quite few in real production servers, so there is no obvious impact on the volume change after the deployment.

5. RELATED WORK

This section reviews prior research studies related to our work. System monitoring has become a significant research area of the IT industry in the past few years. Commercial products such as IBM Tivoli [2], HP OpenView [1] and Splunk [5] provide system monitoring. Numerous studies [15] [6] [19] [34] [10] [23] focus on monitoring that is critical for a distributed network. A number of studies focused on the analysis of historical events with the goal of improving the understanding of system behaviors. A significant amount of work was done on analysis of system log files and monitoring events. Another area of interest is the identification of actionable patterns of events and misses, or false negatives, by the monitoring system. False negatives are indications of a problem in the monitoring software configuration, wherein a faulty state of the system does not cause monitoring alerts.

Table 4: Sampled Rules for Account2 with Testing Data Ratio = 0.3

Situation	Rule Condition	$wait_p$	FP_p	FD_p
cpu_xuxw_std	N/A	355 min	7093	5
monlog_3ntw_std	current_size_64 >= 0 **and** record_count >= 737161	80 min	23	0
svc_3ntw_vsa_std	binary_path = R:\IBMTEMP\VSA\VSASvc_Cli.exs	30 min	27	0
fss_xuxw_std	inodes_used <= 1616 **and** mount_point_u = /logs	285 min	12	2
fss_xuxw_std	inodes_used <= 1616 **and** sub_origin = /logs	285 min	12	2

Table 5: False Negative Tickets

Situation	**Ticket**
dsp_3ntc_std	Please clear space from E drive xxxx-fa-ntfwwfdb Please clear space from E drive xxxx-fa-ntfwwfdb.it is having 2 MB free...
fss_rlzc_std	/opt file system is is almost full on us97udb010ampsb Hi Team@/opt file system is almost full. Please clear some space /home/dbasso>df -h /optFilesystem...
svc_3ntc_std	RFS101681 E2 Frontier all RecAdmin services are down Frontier RecAdmin services are not running on the batch server Kindly logon to the server : xxx.xxx.155.183/xxx ...
dboffln_3oqc_std	DB2 is not connectable from xxxxx Hi Team@Can you please look into why we are unable to connect to Porfolio XRef DB.Server : xxxx12DB Instance : sec_mastId : ipxrbtchWhile...
dboffln_3oqc_std	Unable to login to DB server Hi Team@We had raised a request 131443 for access on the E1 and E2 serversE1 - Full access@ to read/write/execute programs Hostname Server xxxxx xxx.xxx.147.194

Table 6: Accuracy of the word-match method

Situation	**Words**	**Precision**	**Recall**	**F1Score**
File System Space	space,file	0.0341	0.8	0.0654
Disk Space	space,drive	0.1477	0.9565	0.2558
Service Not Available	service,down	0.1941	0.75	0.3084
Router/Switch Down	router	0.6581	0.7404	0.6968

Network monitoring is used to check the "health" of communications by inspecting data transmission flow, sniffing data packets, analyzing bandwidth, etc. [15] [6] [19] [34] [10] [23]. It is able to detect node failures, network intrusions, or other abnormal situations in the distributed system. The main difference between the network monitoring and framework we consider is the monitored target, which can be any component or subsystem of the system, hardware (such as CPU, hard disk) or software (such as a database engine, or web server). Only the system administrators, who are working the monitored server, can determine whether an alert is real or false. This is why we incorporate ticket resolutions, which record how system administrators resolve those alerts using our solution.

A significant amount of work in data mining has been done to identify actionable patterns of events. See example, [13], [21], [16], [32]. Different types of patterns, such as (partially) periodic patterns, event bursts, and mutually dependent patterns were introduced to describe system management events. Efficient algorithms were developed to find and interpret such patterns. Our work is based on the part of an event processing workflow that takes into account the human processing of the tickets. This allowed us to identify non-actionable patterns and misses of the monitoring system configuration with significant precision. In the event processing workflow, false positive events are transformed into false positive tickets. Identification of false positive events makes it possible to significantly reduce the number of false positive tickets. The translation of the actionable patterns to enterprise software rules is considered in [11] and [22]. We implemented our findings as a component prototype for Enterprise Console. The framework obtains information about user preferences and SLA, mines events and suggests monitoring conditions as well as duration parameters. A new set of rules is advised for mined false negatives.

Dealing with false negatives, or misses of the system, usually includes the consideration of additional source of data. In our case, this additional source is ticketing data. As a source of information, it is difficult data to process, because there are no supporting standards or structure, and ticketing records are usually byproducts of the SA work, which are mainly incomplete and unfinished. An additional difficulty is that false negatives are rare and unbalanced due to the fact that historically tested and tuned configurations of the monitoring systems are used. Methods of dealing with unbalanced data was considered for example in [9].

To process the ticketing and logged data, specialized parsers were created to parse and transform applications and information system operation logs. Usually, logs and tickets are semi-structured, containing both structured (e.g., log entry prefixes and timestamp) and unstructured text (e.g., exception, error or warning descriptions, and display of application state). The parsers transform them into relational or extensible (XML like semi-structured) formats and can operate in an offline (like [29], [30], [7], [35]) or online, streaming regime (like [12]). In our work, parsers are used to translate monitoring events and ticketing data into attribute-value pairs convenient for further analysis. Unlike existing work, however, we also include the analysis of ticket resolution descriptions for identifying real tickets where a non-trivial amount of work has been done. Such information was used to tag monitoring events as real alerts or false alerts.

Parameter tuning in log pattern mining is studied in [15], [6]. Usually mining parameters describe how strongly elements of the pattern are interconnected or correlated (e.g., confidence), and what percentage of the data stream should be covered (e.g., support). Tuning up parameters is a delicate process. Parameters that we tune are the percentage of false alerts covered and the number of events covered.

Discovering time-related patterns from system logs is considered in [18], [33]. In our study, the duration time of a pattern depends on a couple of factors such as actual delay time and acceptable SLA thresholds. While the distribution of recognized non-actionable patterns depends only on historical data, we take the delay tolerance of a customer as additional input.

6. CONCLUSION

This paper presents an integrated framework to optimize the automatic system monitoring in large IT infrastructures. By combing the system event data and ticket data from IT service centers, this framework reduces the number of false positive (non-actionable) alerts and the number of false negative (missing) alerts for the automatic monitoring system. It minimizes the cost of providing effective and reliable

means for problem detection. The integrated framework has been implemented as a system in the IBM IT service management platform and deployed in several IBM service centers. This system is used periodically to refine and adjust monitoring situations after a system has gone through a change, thus helping to enhance the overall reliability in IT service management.

7. REFERENCES

[1] HP OpenView : Network and Systems Management Products. `http://www8.hp.com/us/en/software/enterprise-software.html`.

[2] IBM Tivoli : Integrated Service Management software. `http://www-01.ibm.com/software/tivoli/`.

[3] IBM Tivoli Monitoring. `http://www-01.ibm.com/software/tivoli/products/monitor/`.

[4] ITIL. `http://www.itil-officialsite.com`.

[5] Splunk: A commercial machine data managment engine. `http://www.splunk.com/`.

[6] S. Agrawal, S. Deb, K. V. M. Naidu, and R. Rastogi. Efficient detection of distributed constraint violations. In *Proceedings of ICDE*, pages 1320–1324, Istanbul, Turkey, 2007.

[7] M. Aharon, G. Barash, I. Cohen, and E. Mordechai. One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In *Proceedings of ECML PKDD*, pages 227–243, 2009.

[8] L. A. Castillo, P. D. Mahaffey, and J. P. Bascle. Apparatus and method for monitoring objects in a network and automatically validating events relating to the objects. U.S. Patent, Dec. 2008. US 7,469,287 B1.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[10] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *ACM SIGCOMM Conference*, pages 137–148, 2003.

[11] J. Gao, G. Jiang, H. Chen, and J. Han. Modeling probabilistic measurement correlations for problem determination in large-scale distributed systems. In *Proceedings of ICDCS*, pages 623–630, 2009.

[12] G. Grabarnik, A. Salahshour, B. Subramanian, and S. Ma. Generic adapter logging toolkit. In *Proceedings of the international conference on Autonomic Computing*, pages 308–309. IEEE, 2004.

[13] J. L. Hellerstein, S. Ma, and C.-S. Perng. Discovering actionable patterns in event data. *IBM Systems Journal*, 43(3):475–493, 2002.

[14] E. Hoke, J. Sun, and C. Faloutsos. InteMon: Intelligent system monitoring on large clusters. In *Proceedings of VLDB*, pages 1239–1242, 2006.

[15] S. R. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla. Efficient constraint monitoring using adaptive thresholds. In *Proceedings of ICDE*, pages 526–535, Cancun, Mexico, 2008.

[16] J. Kiernan and E. Terzi. Constructing comprehensive summaries of large event sequences. In *Proceedings of ACM KDD*, pages 417–425, Las Vegas, Nevada, USA, August 2008.

[17] J. Li. Robust rule-based prediction. *IEEE Trans. Knowl. Data Eng. (TKDE)*, 18(8), 2006.

[18] T. Li, F. Liang, S. Ma, and W. Peng. An integrated framework on mining logs files for computing system management. In *Proceedings of ACM KDD*, pages 776–781, August 2005.

[19] S. Mccanne and V. Jacobson. The bsd packet filter: A new architecture for user-level packet capture. In *USENIX Technical Conference*, pages 259–270, 1993.

[20] M. J. Pazzani, C. J. Merz, P. M. Murphy, K. Ali, T. Hume, and C. Brunk. Reducing misclassification costs. In *Proceedings of ICML*, pages 217–225, New Brunswick, NJ, USA, July 1994.

[21] W. Peng, C. Perng, T. Li, and H. Wang. Event summarization for system management. In *Proceedings of ACM KDD*, pages 1028–1032, 2007.

[22] C. Perng, D. Thoenen, G. Grabarnik, S. Ma, and J. Hellerstein. Data-driven validation, completion and construction of event relationship networks. In *Proceedings of ACM SIGKDD*, pages 729–734, 2003.

[23] M. J. Ranum, K. Landfield, M. T. Stolarchuk, M. Sienkiewicz, A. Lambeth, and E. Wall. Implementing a generalized tool for network monitoring. In *USENIX Systems Administration Conference*, pages 1–8, 1997.

[24] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21:164–206, 2003.

[25] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1984.

[26] V. Sindhwani and P. Melville. Document-word co-regularization for semi-supervised sentiment analysis. In *Proceedings of ICDM*, pages 1025–1030, 2008.

[27] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of ACM SIGMOD*, pages 1–12, 1996.

[28] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2005.

[29] L. Tang and T. Li. LogTree: A framework for generating system events from raw textual logs. In *Proceedings of ICDM*, pages 491–500, December 2010.

[30] L. Tang, T. Li, and C. Perng. LogSig: Generating system events from raw textual logs. In *Proceedings of ACM CIKM*, 2011.

[31] L. Tang, T. Li, F. Pinel, L. Shwartz, and G. Grabarnik. Optimizing system monitoring configurations for non-actionable alerts. In *Proceedings of IEEE/IFIP NOMS*, pages 34–42, 2012.

[32] L. Tang, T. Li, and L. Shwartz. Discovering lag intervals for temporal dependencies. In *Proceedings of ACM SIGKDD*, pages 633–641, 2012.

[33] P. Wang, H. Wang, M. Liu, and W. Wang. An algorithmic approach to event summarization. In *Proceedings of ACM SIGMOD*, pages 183–194, Indianapolis, Indiana, USA, June 2010.

[34] K. Xu, Z.-L. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *ACM SIGCOMM Conference*, pages 169–180, 2005.

[35] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Experience mining Google's production console logs. *Proceedings of SLAML*, 2010.

[36] X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proceedings of SDM*, 2003.