Software Analytics for Incident Management of Online Services: An Experience Report

Jian-Guang Lou, Qingwei Lin, Rui Ding,
Qiang Fu, Dongmei Zhang
Microsoft Research Asia, Beijing, P. R. China
{jlou, qlin, juding, qifu, dongmeiz}@microsoft.com

Tao Xie
University of Illinois at Urbana-Champaign
Urbana, IL, USA
taoxie@illinois.edu

Abstract **- As online services become more and more popular, incident management has become a critical task that aims to minimize the service downtime and to ensure high quality of the provided services. In practice, incident management is conducted through analyzing a huge amount of monitoring data collected at runtime of a service. Such data-driven incident management faces several significant challenges such as the large data scale, complex problem space, and incomplete knowledge. To address these challenges, we carried out two-year software-analytics research where we designed a set of novel data-driven techniques and developed an industrial system called the Service Analysis Studio (SAS) targeting real scenarios in a large-scale online service of Microsoft. SAS has been deployed to worldwide product datacenters and widely used by on-call engineers for incident management. This paper shares our experience about using software analytics to solve engineers' pain points in incident management, the developed data-analysis techniques, and the lessons learned from the process of research development and technology transfer.**

Index Terms **-- Online service, service incident diagnosis, incident management.**

I. INTRODUCTION

Software industry has been under the movement from traditional shrink-wrapped software to online services (e.g., from shrink-wrapped Microsoft Office to online Microsoft Office 365). Online service systems such as online banking systems and e-commerce systems have been increasingly popular and important in our society.

Online services differ from traditional shrink-wrapped software in various aspects, including their characteristics of continuously running along with aiming for 24x7 availability of services. However, during operation of an online service, there can be a live-site service incident: an unplanned interruption/outage to the service or degradation in the quality of the service. Such incident can lead to huge economic loss or other serious consequences. For example, the estimated average cost of one hour's service downtime for Amazon.com is $180,000 [19]. Online services such as Amazon, Google, and Citrix have experienced live-site outages during the past couple of years [1][16].

Therefore, service providers have invested great efforts on service-quality management to minimize the service downtime and to ensure high quality of the provided services. For example, an important aspect of service-quality management is incident management [3]: once a service incident occurs, the service provider should take actions immediately to diagnose the incident and restore the service as soon as possible. Such incident management needs to be efficient and effective in order to ensure high availability and reliability of the services.

A typical procedure of incident management in practice (e.g., at Microsoft and other service-provider companies) goes as follow. When the service monitoring system detects a service violation, the system automatically sends out an alert and makes a phone call to a set of On-Call Engineers (OCEs) to trigger the investigation on the incident in order to restore the service as soon as possible. Given an incident, OCEs need to understand what the problem is and how to resolve it. In ideal cases, OCEs can identify the root cause of the incident and fix it quickly. However, in most cases, OCEs are unable to identify or fix root causes within a short time. For example, it usually needs to take a long delay to fix the root causes (e.g., code defects), to conduct regression testing of the new build, and to re-deploy it to datacenters. Such whole process causes much delay before the service can be recovered and continue to serve the users. Thus, in order to recover the service as soon as possible, a common practice is to restore the service by identifying a temporary workaround solution (such as restarting a server component) to restore the service. Then after service restoration, identifying and fixing the underlying root cause for the incident can be conducted via offline postmortem analysis.

Incident management of an online service differs from the debugging of shrink-wrapped software in two main aspects. First, incident management requires the service provider to take actions immediately to resolve the incident, as the cost of each hour's service downtime is high [19]. Second, due to the requirement of continuously running, unlike shrink-wrapped software, when an incident occurs in an online service, it is usually impractical to attach a debugger to the service to conduct diagnosis.

In practice, incident management of an online service heavily depends on monitoring data collected at runtime of the service such as service-level logs, performance counters, and machine/process/service-level events. Such monitoring data typically contains information to reflect the runtime state and behavior of the service. Based on the monitoring data, service incidents are timely detected in the form of service anomalies and quality issues. To collect such data, the service system is instrumented with an instrumentation infrastructure (e.g., the System Center Operations Manager [4]) and continuously monitored. For example, a service system at Microsoft under our study generates about 12 billion lines of log messages each day for incident management.

Given that incident management of online services is data-driven by nature, it is a perfect target problem for software-analytics research. Software analytics [25][26][27] has recently emerged as a promising and rapidly growing research area for data-driven software engineering, with strong emphasis on industrial practice. In particular, software analytics is to utilize data-driven approaches to enable software practitioners to perform data exploration and analysis to obtain insightful and actionable information for completing various tasks around software systems, software users, and software development process. In software analytics, a great amount of work on successful technology transfer has already been conducted at the Software Analytics group at Microsoft Research Asia, e.g., performance debugging in the large [14], clone detection [10].

In this paper, we formulate incident management of an online service as a software-analytics problem [25][27], which can be tackled with phases of task definition, data preparation, analytic-technology development, and deployment and feedback gathering. The task of incident management is defined to consist of two parts: (1) incident investigation and diagnosis, and (2) healing suggestion for actions taken to recover the service as soon as possible. Data preparation aims to collect monitoring data of the service for incident management. Analytic-technology development is to develop an incident-management system by formulating problems and developing algorithms and systems to explore, understand, and get insights from the data. During deployment and feedback gathering, feedback is gathered on how practitioners use the developed system in their routine daily work, and then it is used to guide further improvement of the system under consideration.

By tackling incident management with software analytics, we have developed the first industrial system for incident management of online services and deployed the system within Microsoft. Producing high impact on industrial practices, our system is being used continuously since 2011 by Microsoft engineers for effective and efficient incident management of Service X (we use an alias here due to confidentiality). Our system for Service X incorporates various novel techniques that we have developed for addressing significant real-world challenges of incident management posed in large-scale online services.

Throughout the two-year process of conducting software-analytics research for producing such high-impact system, we have gained a set of lessons learned, which are valuable for us and for the broad community of software engineering to carry out successful technology transfer and adoption. We started the project on data-driven performance analysis for online services in 2010. It took us two years to conduct algorithm research, build the diagnosis system, and make the system indispensable for the service-engineering team of Service X. In this paper, we share our lessons learned in our project through three dimensions: solving real problems in practice, improving performance and usability of the developed system, and investing in system building.

In summary, this paper makes the following main contributions:

- The formulation of incident management of online services as a software-analytics problem, which can be tackled with phases of task definition, data preparation, analytic-technology development, and deployment and feedback gathering.
- The first industrial system developed and deployed for incident management of Service X (a geographically distributed, web-based service serving hundreds of millions of users) and various novel techniques incorporated in the system for addressing significant real-world challenges.
- A set of lessons learned (throughout the two-year process of conducting software-analytics research for producing such high-impact system), which are valuable for us and for the broad community of software engineering to carry out successful technology transfer and adoption.

The rest of the paper is organized as follows. Section II introduces Service X. Section III presents our formulation of service-incident management as a software-analytics problem. Section IV presents the resulting SAS system and its techniques. Section V discusses related work. Section VI presents the lessons learned, and Section VII concludes the paper.

II. BACKGROUND OF SERVICE X

Service X is a web-based, external-facing Microsoft service. Similar to other online services, Service X is expected to provide high-quality service on 24x7 basis. During a certain period of time when running the service, the Service X teams were facing great challenges in improving the effectiveness and efficiency of their incident management in order to provide high-quality service. We set up our goals to help the Service X teams solve the incident-management problems. In addition, because the architecture of Service X is representative of typical multi-layer online services, we expect that our techniques designed for Service X are general enough to be applied to other similar online services.

A. Overview of Service X

Service X is a geographically distributed, web-based service serving millions of users simultaneously. Figure 1 illustrates the architecture of Service X. There are more than 10 different types of server roles in the system, including web front end servers, application servers serving various application services, and database servers, etc.

In order to provide high-quality service, Service X is instrumented at development time and continuously monitored at runtime. The monitoring data collected for Service X mainly consists of three types: performance counters, events from the underlying Windows operating system, and the logs created by various components of Service X. The monitoring data is used to detect service incidents in the form of availability or latency issues. When a service incident is detected, the monitoring system of Service X would automatically send an alert email and make a phone call to a team of service engineers, namely On-Call Engineers (OCEs), to trigger the investigation of the incident. The monitoring data would then be used by the OCEs to diagnose the incident and help decide on

what actions to take in order to restore Service X as quickly as possible.

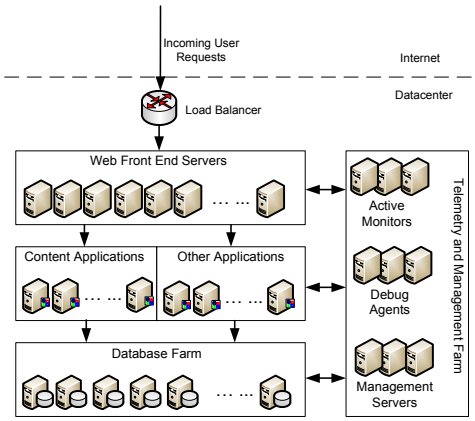


Figure 1. System overview of Service X

B. Pain points and challenges

Incident management is a challenging task because OCEs are under great time pressure to restore the service. From the communication with the OCEs, we learned the following challenges faced by them in incident management. Although these challenges are from the OCEs of Service X, such challenges are general to engineers of other online services because of high resemblance of Service X to general online services.

Large-volume and irrelevant data. The monitoring data is the primary sources for OCEs to diagnose a service incident and identify the restoration actions. The volume of the monitoring data is huge due to the large scale of the Service X system. For example, currently, about 12 billion log entries are generated each day by various service components. The amount will increase rapidly as the number of users increases and/or the number of user requests increases. In addition, most of the monitoring data is irrelevant to a particular incident. From the diagnosis perspective, there is a huge amount of irrelevant data. OCEs would have to manually sift through the huge amount of monitoring data in order to identify the portions relevant to the underlying incident. Sometimes OCEs would not even have a clue on where to start. This process is just like finding a needle in a haystack.

Highly complex problem space. There are many potential causes that may incur a service incident, such as hardware failures, networking issues, resource competition, code defects, and configurations. In general, various types of monitoring data need to be collected in order to gather enough information that reflects the symptoms of complex causes, because each type of data usually reflects only certain aspects of the service system. For example, performance counters are helpful when diagnosing service issues caused by resource competition. In the case of Service X, as aforementioned, performance counters, system events, and logs are collected as monitoring data. When working on incident management, OCEs would not only need to manually analyze each type of the monitoring data, but also need to be able to correlate different types of data in order to obtain thorough understanding of the service incident. It is inefficient to manually look for answers in such a highly complex problem space.

Incomplete and disaggregated knowledge. Diagnosing service incidents often needs decent knowledge about the service system. However, in practice, such kind of knowledge is often not well organized or documented. A large-scale online service system usually consists of many components. These components are usually developed by different teams. Very few engineers have detailed knowledge about the entire system. Therefore, the experts of the service system usually become the bottleneck for incident management. We indeed have such observation with the Service X teams. In addition, from the communication with the OCEs of Service X, we also learned that there was no systematic mechanism for them to share knowledge learnt from past service incidents. Although each incident was recorded in a database, there was no support on reusing the information of those incidents except manual work. Due to the constraints of incomplete and disaggregated knowledge, service engineers are often slow to resolve service incidents, resulting in long Mean Time to Restore (MTTR) for the service.

In the case of Service X, the service engineers used to suffer from the aforementioned pain points during a certain period of time when they were running Service X. Their MTTR was about 2 hours during that time, and 90% of the time was spent on manual inspection of the monitoring data in order to diagnose problems and identify the right restoration actions.

III. INCIDENT MANAGEMENT AS SOFTWARE ANALYTICS

As discussed in Section II, there are a set of practical challenges in the incident management of Service X. Because the core problem is how to effectively and efficiently analyze the huge amount of monitoring data in order to come up with the diagnosis and restoration actions, we formulate the incident-management problem as a software-analytics problem. We utilized the four-step approach of developing software analytics projects [25][27] to define the objectives of our project, conduct data collection, develop analytics techniques and an analysis system leveraging those techniques, as well as deploying the analysis system and getting feedback. The analysis system that we developed is named as the Service Analysis Studio (SAS).

In this section, we present the four steps of developing SAS. We first define the objectives of SAS. Then we illustrate the different types of monitoring data used for analyzing service incidents. We further discuss the four primary analysis techniques that we developed. Finally, we discuss the user interface design of SAS and collection of user feedback in real deployment.

A. Objectives

We defined four main objectives for SAS, in order to help the OCEs of Service X to overcome the practical challenges in their incident-management effort.

Automating analysis. SAS should have the capability to automatically identify the information relevant to the cause of

the incident under investigation from the huge amount of monitoring data. The identified information should provide insightful clues for OCEs to determine the problematic site of the incident, therefore significantly reducing the investigation effort.

Handling heterogeneity. SAS should be able to analyze the various types of monitoring data collected from different data sources. In the case of Service X, the types of monitoring data included performance counters, system events, and logs generated by different service components. Each data source provides the diagnostic information of Service X from a certain aspect. Different from all previous work [6][8][9][24] that focused on only a single type of data source (e.g., system metrics), SAS aims to provide a comprehensive analysis of the various types of data from all data sources to support the diagnosis of service incidents.

Accumulating knowledge. SAS should provide a mechanism to accumulate and leverage the knowledge about the incidents. Similar to other services in the real world, the same incident of Service X may reoccur due to various reasons. For example, the bug fix for the root cause of the incident has not yet been deployed, a temporary workaround solution stops to take effect, or the service suffers repetitively from high workload and resource competition. Accumulating the knowledge about past incidents can help improve the effectiveness and efficiency of incident management. If OCEs can quickly determine that a new incident is similar to a previous one, then they will be able to quickly restore the service by leveraging the diagnosis effort of the previous one. SAS is targeted to accumulate the knowledge of past incidents by constructing a historical incident repository, and to leverage such knowledge to resolve new incidents.

Supporting human-in-the-loop (HITL). SAS should provide flexible and intuitive user interfaces in order to enable OCEs to effectively and efficiently interact with the analysis results and the monitoring data. The diagnosis of a service incident is a complex decision-making process. Given the complexity and diversity of service incidents, it is too ambitious and not realistic in practice to build and deploy a fully automatic diagnosis system in real production environments. Therefore, rather than making incident management fully automatic, we keep the OCEs in the loop to make decisions on the diagnosis and identification of restoration actions. Meanwhile, we fully utilize the power of data-analysis algorithms to provide as much information as possible to facilitate the decision making of the OCEs.

B. Monitoring Data of Service X

As introduced in previous sections, different types of data are collected in order to monitor the quality of Service X, as well as diagnosing service incidents. In this section, we discuss each type of the monitoring data in detail. We also explain how the quality of Service X is measured and how service incidents are detected.

Detecting incidents during service operation is often based on the Key Performance Indicators (KPI), such as the average request latency and request-failure rate. In the case of Service X, for each user request, the response time is recorded at the service side (as the request duration) along with the HTTP status code (http-status-code) of the response to the request. The http-status-code indicates the returned status of a given web request, e.g., 200 refers to "OK" and 500 refers to "Internal Server Error". The duration indicates the total response time, e.g., duration>10 seconds indicates that the user has experienced very slow response. These two attributes are used to calculate the KPIs for Service X. Each KPI is calculated once per time epoch (i.e., 5 minutes in the system of Service X). For example, for each time epoch, the 95-percentile latency is calculated based on the duration values of all requests within the time epoch. KPI values are monitored to provide an overall description about the health state of Service X from users' perspective. In practice, the values of KPIs are checked against certain specified Service Level Objective (SLO). The SLO is defined to be the acceptable value ranges of KPIs. When Service X is running, if a KPI's value (e.g., average latency) violates the SLO, a KPI violation, i.e., service incident, is detected, and alerts are sent out to notify that the service is in a SLO-violation state. The diagnosis of a service incident is to find out the problematic site that causes the service to violate the SLO.

Besides KPIs, performance counters and system events, which are collectively named as system metrics, are also collected for the diagnosis purpose. System metrics record the measurement results of the system, including the resource usage of processes and machines (e.g., the CPU utilization, disk queue lengths, and I/O operation rate), request workload (e.g., the number of requests), SQL-related metrics (e.g., the average SQL lock waiting time), and application-specific metrics (e.g., the cache hit ratio, the number of throttled requests). These metrics are collected via OS facilities (e.g., Windows Management Instrumentation (WMI)) and stored in a SQL database. Similar to KPIs, each system metric is also aggregated over time epochs. For example, two metric values are calculated for the CPU-usage metric over an epoch: the average (or median) value and the maximum value of CPU usage within the epoch. There are more than 1200 different types of metrics collected in Service X.

Another important type of data collected in Service X is transactional logs. Transactional logs are generated during system execution, and they record detailed information about the system's runtime behaviors when processing user requests. Each log entry contains the following fields: the timestamp, request ID (TxID), event ID, and detailed text message.

- A request ID is a Global Unique Identifier (GUID) representing a request instance. Service X can serve multiple requests simultaneously using concurrent threads. These threads write log entries to the same file as they execute, resulting in a log file with interleaving entries of different requests. The log entries can be grouped into different sequences using request IDs. In this way, each group represents the log sequence produced when serving a particular request.
- An event ID is a unique identifier representing an event-logging statement in the source code. The event ID in a log entry indicates which logging statement prints out this

entry. The event ID bridges the logs with the source code – given a log sequence represented by a request ID, the execution path of modules and functions in the source code can be identified. Usually, different types of requests generate different log sequences due to different program logics. Sometimes the same type of requests can also generate different log sequences due to different input and configuration values, etc.

After data preparation, we need to design a set of data-driven analysis techniques targeting at the real scenarios in Service X. These techniques can automatically extract the information from the monitoring data and guide OCEs to find out the problematic site of an incident.

IV. TECHNIQUES IN SAS

A set of data-driven techniques for diagnosing service incidents have been developed in SAS for incident diagnosis in Service X. Each of these techniques targets at a specific scenario and a certain type of data. In this section, we briefly go through some analysis techniques designed for different types of data, along with the SAS user interfaces and deployment-time feedback.

Because poor predictions are produced by just directly applying standard classification algorithms or state-of-the-art information-retrieval techniques without considering characteristics of logs in our scenario [11][12], we designed and extended our techniques based on carefully considering domain-specific characteristics of software-generated data to achieve satisfying performance.

A. Identification of Incident Beacons from System Metrics

When engineers diagnose incidents of online services, they usually start from hunting for a small subset of system metrics that are symptoms incurred by the causes of the incidents. We name such kind of metrics as service-incident beacons. A service-incident beacon is formed from a combination of metrics with unusual values that produce a symptom. It could help directly pinpoint the potential incident causes or could provide intermediately useful information leading engineers to locate the causes. For example, when a blocking and resource-intensive SQL query blocks the execution of other queries accessing the same table, symptoms can be observed on monitoring data: the waiting time on the SQL-inducing lock becomes longer, and the event "SQL query time out failure" is triggered. Such metrics can be considered as incident beacons. There are more than 1200 system metrics in Service X. We developed an analysis technique that helped OCEs effectively and efficiently identify service-incident beacons from such huge number of system metrics.

Our analysis technique consists of three steps. First, using anomaly detection, we discretize the values of system metrics to indicate normal or abnormal states of those metrics. The reason is that a service-incident beacon often has exceptionally high or low values that are significantly out of its normal value range during the period of the incident. Second, we apply correlation analysis to identify incident beacons from suspicious metrics using the historical monitoring data. In particular, with the discretized metric values and the SLO states (indicating whether the SLO is violated or not) of a KPI in each epoch, we mine all the possible association rules between the abnormal metrics and the service violations by leveraging an algorithm for mining Class Association Rules (CARs) [17]. These mined CARs are stored as incident-beacon candidates for the diagnosis purpose. Third, given a newly detected service incident and its corresponding metrics and KPIs during the time period of the incident, we calculate the log likelihood for each CAR candidate obtained in Step 2 to assess how likely it is related to the underlying service incident. The metrics involved in the CARs with top rankings are provided as service-incident beacons to the OCEs. The technical details of our analysis technique can be found elsewhere [12].

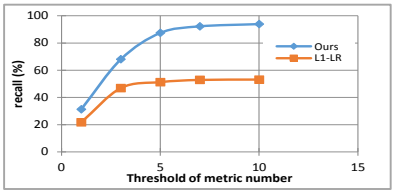


Figure 2. The recall results

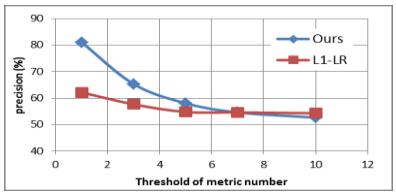


Figure 3. The precision results

We tested some state-of-the-art algorithms proposed to solve similar problems [6][8][9], found that they did not work well for Service X because of the following two main characteristics of incidents of Service X, and then designed our own analysis technique to deal with such characteristics. First, most incidents of Service X last less than 2 hours. Each incident contains only a small number of epochs. When a model is learned for each incident, the previously proposed learning algorithms [6][8][9] would suffer from the over-fitting problem due to the insufficient amount of training data. Our technique reduces the chance of over-fitting because incident beacons are selected from the candidates that are significant rules mined out of the entire historical data set. Second, in practice, a false negative (i.e., a real incident beacon not being reported) can often incur high investigation cost for OCEs, because OCEs would have to go through all the metrics to find the relevant ones. Unlike classification-based techniques that identify a single model [6][8][9] for each incident, our CAR-mining technique can discover all rules that satisfy given requirements including the minimal support, confidence, and lift values. Therefore, our technique can help reduce the false-negative ratio when there are coupling effects [12] in the underlying incidents. The profound differences between classification and association-rule mining [13] can help illustrate why a mining-based technique works for Service X.

We evaluated our analysis technique using real data of Service X. For example, on a data set of 36 incidents, with

nearly the same precision, our technique achieved a high recall (~90%) compared to the recall of ~60% obtained using L1-Logistic Regression (in short as L1-LR, an algorithm in state-of-the-art research [9]). Figures 2 and 3 show the recall and precision results, respectively, as we change the threshold of the number of selected metrics from 1 to 10. We can observe that our technique can achieve better recall and precision in all cases than the technique of L1-LR. In practice, a threshold of 5 or 6 is a good choice. The characteristics of incidents of Service X, such as the short-period violation and coupling effect, are common among many other online services. Therefore, our analysis technique can also be applied to other online services.

B. Mining Suspicious Execution Patterns

Besides system metrics, the transactional logs also provide rich information for diagnosing service incidents. When scanning through the logs, OCEs usually look for a set of log events that show up together in the log sequences of failed requests but not in the ones of the succeeded requests. Such a set of log events are named as suspicious execution patterns. A suspicious execution pattern could be very simple, e.g., an error message that indicates a specific fault in the execution. It could also be a combination of log events of several operations. For example, a normal execution path looks like {task start, user login, cookie validation success, access resource R, do the job, logout}. In contrast, a failed execution path may look like {task start, user login, cookie not found, security token rebuild, access resource R error}. The failure occurred because resource R cannot recognize the new security token when the old cookie was lost. The code branch reflected by {cookie not found, Security token rebuild, access resource X error} indicates a suspicious execution pattern.

As discussed in previous sections, a huge number of logs are generated at any time when Service X is running. It is critical to automatically identify suspicious execution patterns in order to free OCEs from manually scanning the logs. We propose a mining-based technique to automatically identify suspicious execution patterns. The basic idea behind our technique is that, given a set of logs for failed requests and succeeded requests, respectively, execution patterns shared by more failed executions and fewer succeeded executions are more suspicious than others. The details of our technique can be found elsewhere [11]. Our technique mainly consists of two steps.

First, we mine execution patterns by modeling the trunk/branch relations of program-execution paths with a Formal Concept Analysis (FCA) [2] technique. Given a set of transactional logs, we treat each request as an object, the set of events (corresponding to this request) as attributes, and then we apply FCA to obtain a lattice graph. Each node in the graph is a concept. Each concept, denoted as c, contains two elements: intent and extent. $Int(c)$ (denoting the intent of concept c) is an event set, and Ext(c) (denoting the extent of concept c) is a request set. In the graph, each parent concept contains the common path of its children, and each child concept contains a different branch structure in code paths. Then, we further extract a complementary set $\Delta Es = Int(c) \backslash Int(p)$ (the log events that are in node c but not in node p) for every parent-child node pair (p, c) in the graph. All extracted complementary sets ΔEs are stored as candidates of suspicious execution patterns for further evaluation.

Second, we use a score named as Delta Mutual Information (DMI) to measure the suspicious level of each execution pattern. DMI is defined as $DMI(\Delta Es) = M(X_c, Y) - M(X_p, Y)$, where $M(X_c, Y)$ and $M(X_p, Y)$ represent mutual information defined on X_c (a Boolean random variable defined on concept c, with the variable value as 1 if the request belongs to $Ext(c)$ and as 0 otherwise) and Y (the fail/success status of a given request, e.g., 1 if the request is failed). Theoretical analysis has shown that DMI can properly measure the contribution of ΔEs for failure correlation [11]. By walking through all edges in the lattice graph, we select all ΔEs as suspicious execution patterns if each of them has a large DMI value, and then present them to OCEs for diagnosis.

In the practice of Service X, several patterns appeared in incidents in long term, such as ones related to SQL timeout or user-authentication rejection. Some patterns were live in short term, specific to some versions of software, or improper configurations; these patterns disappeared after software upgrade.

C. Detection of Malfunctioned Role Instance

In addition to analyzing the system metrics and transactional logs, based on the characteristics of the system architecture of online services, we also developed a statistics-based technique to help with incident management.

As discussed in Section II, usually there are multiple server roles in a large-scale online service system, e.g., front end server and SQL server. There are often a number of instances for each role running on different servers, under the control of a load balancer that distributes the workload among the peer instances. The configurations of these peer servers with the same role are usually homogeneous for simplicity and robustness. Therefore, when the service is at a healthy state, different instances of the same role should have similar behaviors. If the behavior of one instance deviates far from its peer instances, then this instance is likely to act in an abnormal state. Such behavioral differences can help us quickly detect the instances of malfunctioned server roles.

The detection algorithm consists of two steps. First, a metric (denoted as V) reflecting the health state of a role is selected, and its values are monitored for each role instance. For a specific role, we calculate its values across all the instances in the time epoch of investigation, and learn a probabilistic model from the calculated metric values. In SAS, for simplicity, we use Gaussian distribution $N(\mu, \sigma)$ to model the metric. The parameters (μ, σ) are estimated using a robust estimation method to reduce the interference of outliers:

$$\begin{cases} \mu = median(V) \\ \sigma = 1.48 \times median(|v - \mu| \forall v \in V) \end{cases}$$

Second, we identify the role instances whose corresponding metric values are far from the distribution $N(\mu, \sigma)$.

In SAS, we use the preceding technique to detect the malfunctioned instances of three roles: the front end server, application server, and SQL server. This technique is simple,

and yet we have found it highly effective in real practice. It can often locate the problematic servers with high accuracy, thus effectively narrowing down the investigation scope for OCEs. This technique is not limited to Service X, being general and applicable to common online services.

D. Leveraging Previous Effort for Recurrent Incidents

Similar incidents may reoccur due to reasons discussed in Section III-B. Therefore, leveraging the knowledge from past incidents can help improve the effectiveness and efficiency of incident management. The key here is to design a technique that automatically retrieves the past incidents similar to the new one, and then proposes a potential restoration action based on the past solutions.

Incident retrieval. There is rich information associated with each service incident, e.g., timestamp, monitoring data, and text describing the symptoms, diagnosis, taken actions, and results, etc. The monitoring data is the most important because it faithfully reflects the states of the service system during the incident. Therefore, we use the monitoring information to derive signatures to represent the incidents for the retrieval purpose. Using the technique discussed in Section IV, we mine out the suspicious execution patterns in transactional logs, and use such patterns as signatures for each incident.

Then we define a similarity metric to compare a new incident to past ones. We treat each incident as a document, each execution pattern as a term, and the corresponding DMI score as the weight of the term. We then use the Generalized Vector Space Model [22] to calculate the similarity of two incidents.

Table 1. Healing actions

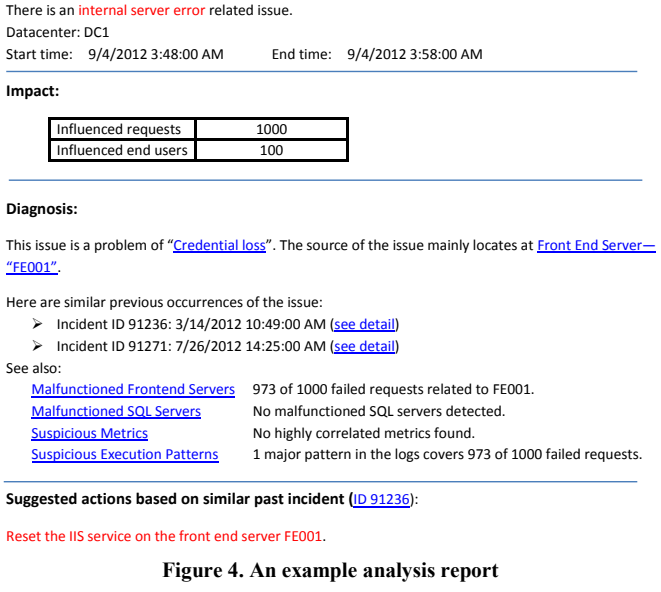
verb	*target*		*verb*	*target*
recycle	App-pool		re-image	WFE
restart	IIS or Service		rotate	WFE
reboot	WFE		rotate	SQL
reboot	SQL		patch	WFE
reset	DB		patch	SQL

Healing-action adaptation. According to our empirical study of healing actions in the incident repository, we find that most healing actions can be formatted as a tuple <verb, target, location>, where "verb" denotes an action and "target" denotes a component or service. Table 1 shows all "verbs" and "targets" in SAS. When we retrieve a similar historical incident, we extract the verb and target from its description text. For example, we extract "reboot" as the verb and "SQL server" as the target from the description "We found few SQL servers with high memory usage and few servers were not able to connect through SSMS. Availability is back up after rebooting these SQL machine SQL32-003". We determine the location using the technique that detects the malfunctioned server role.

Evaluation. We have evaluated the effectiveness of our technique using the "leave-one-out" strategy based on 77 real incidents of Service X. These cases are grouped into 8 categories; we measure the accuracy of our technique's effectiveness in suggesting a correct healing action for each "new issue" (i.e., the one that is left out during "leave-one-out"). Both the two results show that our approach is effective. The average accuracy of top-1 recommendation is 0.90. More detailed results including the ROC curves of our technique can be found elsewhere [11].

E. Usability

As a practical tool, making the analysis results actionable and understandable to OCEs is very important. Otherwise, the tool would not make real impact or be widely used by OCEs.


There is an internal server error related issue.
Datacenter: DC1
Start time: 9/4/2012 3:48:00 AM End time: 9/4/2012 3:58:00 AM

Impact:

Influenced requests	1000
Influenced end users	100

Diagnosis:

This issue is a problem of "Credential loss". The source of the issue mainly locates at Front End Server—"FE001".

Here are similar previous occurrences of the issue:
- Incident ID 91236: 3/14/2012 10:49:00 AM (see detail)
- Incident ID 91271: 7/26/2012 14:25:00 AM (see detail)

See also:

Malfunctioned Frontend Servers	973 of 1000 failed requests related to FE001.
Malfunctioned SQL Servers	No malfunctioned SQL servers detected.
Suspicious Metrics	No highly correlated metrics found.
Suspicious Execution Patterns	1 major pattern in the logs covers 973 of 1000 failed requests.

Suggested actions based on similar past incident (ID 91236):

Reset the IIS service on the front end server FE001.

Figure 4. An example analysis report

One pain point of the OCEs is to sift through a huge amount of monitoring data when working on a service incident. To address such pain point, we defined two design rationales for presenting the analysis results in SAS: conciseness and comprehensiveness. Based on the results generated using different analysis techniques, SAS can automatically compose an analysis report using a predefined decision tree. This report serves as the primary form of presentation for SAS to communicate its analysis results to OCEs. As shown in Figure 5, the report is concise, and yet contains comprehensive information about the underlying incident. The report has three parts. It first provides information on the impact of the incident, e.g., the number of failed user requests and the number of impacted users. Such information helps the OCEs to assess the severity of the incident. The second part of the report provides information for assisting effective diagnosis including the summary of the underlying issue (if found), a list of similar incidents in the past, and links to the detailed diagnosis results of each type of the monitoring data. This report provides an easy and systematic way for service engineers to consume the analysis results, and thus greatly improves the usability of SAS. For example, OCEs can quickly get an overview of the incident and understand what was going on during the period of the incident. They can also obtain detailed information for further investigation through a single mouse click. The third part of the report recommends service-recovery actions adapted from those for similar incidents in the past. For example, in Figure 4, the suggested action is to reset the IIS on a specific front end server.

In addition, we present the results of suspicious execution patterns in an easy-to-understand way in SAS. Many terms in

the machine-learning and data-mining areas are not easily accessible to OCEs. For example, many OCEs are not familiar with execution patterns or FCA. In SAS, we use a UI to highlight the common difference between logs of succeeded and failed requests, and facilitate OCEs to intuitively manipulate the log sequences for understanding the contribution of different log messages to the failure.

F. Deployment and Feedback

SAS was first deployed to the datacenters of Service X worldwide in June 2011. The OCEs of Service X have been using SAS for incident management since then. Now, they heavily depend on SAS. Because of its importance for Service X, we were required to make sure the high availability of SAS. However, in practice, it is very difficult to estimate how much OCE time a tool helps save. In order to assess the impact of SAS in practice, we have instrumented SAS and started to collect its usage data since 2012. The usage data records all the interactions between users and SAS. Based on the usage data, we can answer questions such as "who uses which analysis module at what time on what data?"

According to the usage data from a 6-month study, about 91% of OCEs used SAS to accomplish their incident-management tasks. SAS was used to diagnose about 86% of service incidents. Along with engineers from Service X teams, we investigated whether the analysis results of SAS were useful for diagnosing an incident. The ground truth is set up according to the product-ticketing system. In particular, for each service incident, a ticket is created in the ticketing system to record the detailed information of the diagnosis process of the service incident including symptoms, email threads, diagnosis results, and recovery actions. We use the recorded tickets as the ground truth, and compare them with our analysis results to conduct the evaluation. The results are considered useful if (1) they can directly help locate the cause of the incident; (2) they can locate the malfunctioned component; or (3) they can find out the problematic site to help OCEs to reduce their investigation scope. The data in these 6 months shows that SAS helped diagnose about 76% of the service incidents that SAS was used for.

There are two main reasons why SAS failed to provide useful diagnosis information for the remaining 24% service incidents. First, sources of incident causes were not covered by the monitoring system. For example, in the production environment of Service X, several incidents were caused by a malfunctioned Active Directory (AD) controller. Since no monitoring information was collected on AD servers back then, SAS could not provide useful clues for diagnosis. Second, there are inconsistencies and errors in the transactional logs. Such factor may impact the precision of our incident retrieval algorithm.

In summary, with the techniques of data analysis, SAS tackled challenges in practice, and it helped OCEs of Service X improve their effectiveness and efficiency of incident management. We expect that the analysis techniques and design principals of SAS can be applied to other online services.

V. RELATED WORK

Previous work applies statistical-analysis techniques (i.e., machine learning and data mining) to tackle the scale and complexity challenges in incident management. We discuss related work in three categories.

Incident-beacon identification. Previous work [6][8][9][15][24] mainly focused on finding suspicious system metrics that may be related to the incident under investigation. Given the data of system-SLO states (violation or compliance) and system metrics, Cohen et al. [6][8][15][24] proposed the Tree-Augmented-Network (TAN) approach to deduce a TAN model, which uses a few system metrics to predict system-SLO states. Their approach identifies the metrics used by the deduced TAN model as service-issue beacons. Bodik et al. [6] adapted their approach by adopting a different model, named as L1-Logistic Regression, to identify highly correlated metrics more accurately. However, these previous classification-based approaches [6][8][9][15][24] usually analyze each performance issue one by one, and have a number of limitations (suffering from the over-fitting problem when learning a classifier for a performance issue with short duration, identifying only general symptoms as incident beacons, etc.)[12]. Our techniques in SAS tackle these problems by mining CARs from historical data, and then selecting the best ones from the candidates by matching them with the performance issue under investigation.

Known-incident association. As discussed earlier, associating a newly incoming incident with a previous known incident is very useful in incident management. Yuan et al. [23] used classification techniques to classify system problems into different categories. However, in real practice, classification-based techniques are often not applicable due to lack of labeled samples. In addition, a classification-based technique often cannot check whether an incident is a totally new one or similar to a previous one. Previous work [6][8][9][15][24] retrieved similar previous incidents by defining similarity based on the beacons of incidents (those beacons are used as incident signatures). Another set of research efforts in the area of mining bug repositories is also related to our known-incident association technique. The basic idea of such scenario is to apply web-search techniques on a bug repository where each bug report is considered as a web document. Ashok et al. [5] implemented a search system for similar-bug retrieval to speed up bug fixing based on the natural-language text, dumped traces, and outputs described in the bug reports. Some other work [20][21] uses mining or classification techniques on textual information to cluster or detect duplicate bug reports. These techniques would not be effective in our problem setting because the textual information is a much weaker representation of an incident compared to the monitoring data associated with the incident. Furthermore, the textual information is also incomplete or imprecise [11]. Different from the previous work, we extract incident signatures by analyzing the difference between the logs of failed requests and succeeded requests. In SAS, we go further to provide healing suggestions by leveraging the solutions of previous incidents in the incident repository.

Fault localization. Automated localization of faults/bugs is a major research area in software engineering. Two kinds of localization techniques are used in SAS. The basic idea of our technique for execution-pattern mining is similar to previous work [18][20] in that we all leverage the differences between the logs of failed and succeeded requests. Sun et al. [20] evaluated patterns mined from both correct and incorrect runs to detect duplicate bug reports. Our work uses contrast information to achieve high accuracy of signature generation. Cellier [7] applied FCA to fault localization by using concepts to find interesting clusters. In contrast to these previous techniques on fault localization, our work is motivated by addressing challenges of incident management.

The above-discussed previous work focused on developing techniques for a single type of data sources, and none of them has been deployed to a real-world online service system. In our work, we conducted comprehensive analysis on various monitoring-data types to handle real-world problems, and developed the SAS system, which has been used in real production environments.

VI. LESSONS LEARNED

We started the project on data-driven performance analysis for online services in 2010. It took us two years to conduct algorithm research, build the diagnosis system, and make the system an indispensable system for the engineering team of Service X. In this section, we share some of our experiences and the lessons learned along the way.

A. Solving Real Problems

Solving real problems is one of the key factors to the success of SAS. We did not, however, take the problem-driven approach right at the beginning of the project, and we learned the lesson the hard ways.

When we first knew about the various challenges of Service X, we went on the usual research route looking into the research literature on existing work to understand state-of-the-art techniques in the area. We found that using a machine-learning technique to classify, retrieve, and predict service violations had been an interesting topic, and a classification-based technique was the mainstream solution. We analyzed the pros and cons of several popular classification-based techniques, implemented, and tested them using the real data that we obtained from Service X. However, the results were not satisfactory as discussed in Section IV-A; therefore, we decided to research on this topic in order to improve the recall and precision. We spent a few months along this direction and did get better results later.

We presented to the engineering team from Service X the improvements that we made over the state-of-the-art techniques, and we got feedback such as “interesting”, “good”, and “useful”, as well as questions and comments such as “this technique alone cannot solve our problems”, and “do you guys look at logs as well?”, “How can you help find the root cause?”, etc. It was then when we realized that we missed two important issues. One was that there were other data sources (e.g., service logs) that were important for analyzing service-quality issues but we did not leverage. The other was that the problem that we worked on was important, but it may not be the most important one and it was not the whole problem.

Since we had a real system running, and there were practitioners who faced real challenges and were willing to talk with us, we decided to reset the project and take a problem-driven approach in order to ensure that our research would address the real problems. After a few rounds of communication with the Service X teams, we clearly identified that the top priority for Service X at that time was to greatly reduce the Mean-Time-To-Restore (MTTR), and the primary challenges included dealing with large-scale and heterogeneous data, and leveraging disaggregated knowledge learned from past incidents, etc. Based on these challenges and real-world scenarios, we formulated the incident-management problem of online services as a software-analytics problem, and researched and developed SAS as discussed in the previous sections.

B. Improving Techniques in Practice

Robustness. In a large-scale online-service system, data missing and noise can be common. In the design of techniques, much effort was spent on tuning the underlying algorithms to make them robust in the real scenarios. For example, in order to improve the robustness of our algorithm of malfunctioned-role detection, median values and Medians of Absolute Difference (MAD) are used to estimate the Gaussian parameters. In addition, during the detection stage, we use Bayesian inference by setting a low a-prior failure probability (e.g., 1e-5), which can largely reduce the rate of false positives. In our execution-pattern analysis, each execution pattern is represented by a sub-set of log events rather than a sub-sequence of log events to improve algorithm robustness. Because many distributed-system components serve a single user request collaboratively and their log events may be disordered due to machine-time bias, an algorithm based on execution patterns with temporal sequential events is not robust enough in practice. In addition, our empirical study shows that an event set is an effective abstraction for our problem context (similar observations were also made by Cellier [7]).

Performance. In addition to the enabling algorithms for analyzing the large-scale and heterogeneous data, performance plays an important role in the adoption of SAS in practice. In order to speed up the investigation of service incidents, OCEs need to obtain relevant information as quickly as possible. We paid a lot of attention to ensure high performance when designing and implementing SAS.

In order to enable real-time analysis leveraging historical data, we designed a background service to incrementally process new generated data as it came in, and save the intermediate results for on-demand analysis later on. Our service runs once every 5 minutes, collects the metric data newly generated during the past 5 minutes, calculates the KPIs and metric values from the data, and runs some analysis modules. For example, the first two steps of the technique for identifying incident beacons (see Section IV-A) run as a part of the background service to learn a set of CARs incrementally. The learned CARs are stored as intermediate results, and then are used later for on-demand analysis. On the contrary, the third

step is often run on demand. When an OCE tries to investigate a service incident, he/she often selects the time period of the incident for analysis through the UI of SAS, and lets SAS run the third step of the technique on the metrics for the time period under investigation. Because the third step does not require heavy computation, an OCE can obtain incident beacons immediately.

We also have some special designs in the module of execution-pattern analysis to improve the performance. First, we automatically cache log sequences of a few succeeded requests in a local file, and update the cache every day in the background service. Doing so can help speed up the on-demand analysis by reducing the data-fetching time. Second, during the on-demand analysis, we select a 20-minute time window where the service has the worst performance among the time range of the incident under investigation, and use only the failed requests in the window for analysis to reduce the computational cost of execution-pattern analysis. Such design can largely improve the interactivity of SAS. When an analysis step did take a relatively long time, e.g., a few seconds, related information would be displayed to notify users on what analysis was running along with its progress.

C. *Availability*

Besides the interactivity and the performance, high availability is also very important for a tool designed for online services. When a service incident occurs, OCEs need to use the tool for investigating and resolving the incident as quickly as possible. If the tool is unavailable at that time, OCEs have to spend extra time to restore the tool or to investigate the incident through other ways (e.g., manually inspecting the instrumented data). Therefore, it is important to guarantee the high availability of SAS. In order to improve the robustness of SAS, the background service of SAS is designed to be auto-recoverable from failures. For example, there are a set of check points in the service code. At each check point, we verify the states of the service, and record the states and all intermediate results in files. When the service fails, it is restarted automatically by the operating system, and then, it recovers its states from the latest check-point files. During the past year, we encountered one case: we were called in during a mid-night to fix a SAS issue because OCEs were unable to get the latest analysis report from SAS; such issue was caused by that the account used for SAS was deleted by an operator by mistake.

D. *Investing in System Building*

In addition to conducting algorithm research, we also built the entire SAS system, which was deployed in the datacenters of Service X worldwide. The engineering cost in building such a system was not low. We did not build SAS to its current state all at once. Instead, we took a step-by-step way and added functionalities incrementally. Doing so did not only help pave the way to creating real impact in three main ways (as discussed below), but also helped us maintain the engineering investment within the manageable scope.

First, having a working system helped demonstrate the research value and built trust with the product-team partners. Usually, product teams are under tight schedule and they would not have cycles for "distractions" once they are in the full development mode. In the case of providing online services, they are quite sensitive about deploying systems or tools that consume resources in datacenters and might impact the services in any way. Considering these practical issues, we built SAS v1.0 with the primary functionality of discovering problematic execution patterns associated with the given service incident by analyzing the service logs. This functionality greatly reduced the scope of log investigation from thousands of lines of logs to just tens of lines. We first demonstrated the effectiveness of SAS using historical logs. Then we got the permission to run it within the internal deployment environment of Service X. This step was critical because SAS was made available for the first time to the teams of Service X for troubleshooting, and this step demonstrated that running SAS had negligible impact on Service X. After SAS v1.0 was used to help troubleshoot some incidents, we got the permission to deploy it to the production environment of one datacenter. The success there created the demand of worldwide deployment into all datacenters.

Second, a working system helped us get timely feedback. The feedback allowed us to observe the troubleshooting experiences of service engineers, and it helped us understand how well the service engineers used SAS. At the same time, we instrumented SAS to collect how service engineers used it in the real settings. This data provided quantitative metrics for us to measure the impact of our work. The investigation on why SAS was not used for or could not help with certain incidents could lead to new research problems.

Third, a working system helped us build up credibility and bring in more research opportunities. As more and more teams came to know the success of SAS, they started to come to us with their own problems. Some of them were similar to the challenges of Service X and the others were different. For the similar problems, we could easily reuse the analysis techniques and modules that we built for SAS. Therefore, the engineering investment really paid off, and it would pay off more as components in SAS got (re)used more. The different problems provided new opportunities for us to explore the online service landscape.

VII. CONCLUSION

Incident management has become a critical task for an online service to ensure high quality and reliability of the service. However, incident management faces a number of significant challenges such as the large data scale, complex problem space, and incomplete knowledge. To address these challenges, we developed an industrial system called SAS based on a set of data-driven techniques to improve the effectiveness and efficiency of incident management in a large-scale online service of Microsoft. In this paper, we have shared our experience on incident management for the large-scale online service including the way of using software analytics to solve engineers' pain points in incident management, the resulting industrial system, and the lessons learned from the process of research development and technology transfer.

REFERENCES

[1] "Amazon's S3 cloud service turns into a puff of smoke". In *InformationWeek NewsFilter*, Aug., 2008.

[2] http://en.wikipedia.org/wiki/Formal_concept_analysis

[3] http://en.wikipedia.org/wiki/Incident_management

[4] http://en.wikipedia.org/wiki/system_center_operations_manager

[5] B. Ashok, J. Joy, H.K. Liang, S. K. Rajamani, G. Srinivasa, V. Vangala. "DebugAdvisor: A recommender system for debugging". In *Proc. of ESEC/FSE' 09*, pp. 373-382, 2009.

[6] P. Bodik, M. Goldszmidt, A. Fox, D.B. Woodard, H. Andersen. "Fingerprinting the datacenter: Automated classification of performance crises". In *Proc. of EuroSys' 10*, pp. 111-124, 2010.

[7] P. Cellier. "Formal concept analysis applied to fault localization". In *Proc. ICSE Companion' 08*. pp. 991-994, 2008.

[8] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, J.S. Chase. "Correlating instrumentation data to system states: A building block for automated diagnosis and control". In *Proc. of USENIX OSDI' 04*, pp. 231-244, 2004.

[9] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, A. Fox. "Capturing, indexing, clustering, and retrieving system history". In *Proc. of SOSP' 05*, pp. 105-118, 2005.

[10] Y. Dang, D. Zhang, S. Ge, C. Chu, Y. Qiu, T. Xie, "XIAO: Tuning code clones at hands of engineers in practice". In *Proc. of ACSAC' 12*, pp. 369-378, 2012.

[11] R. Ding, Q. Fu, J.-G. Lou, Q. Lin, D. Zhang, J. Shen, T. Xie, "Healing online service systems via mining historical issue repositories". In *Proc. of ASE' 12*, pp. 318-321, 2012.

[12] Q. Fu, J.-G. Lou, Q. Lin, R. Ding, D. Zhang, Z. Ye, T. Xie, "Performance issue diagnosis for online service systems". In *Proc. of SRDS' 12*, pp. 273-278, 2012.

[13] A. A. Freitas, "Understanding the crucial differences between classification and discovery of association rules - a position paper". In *SIGKDD Exploration*, vol.2, no.1, pp. 65-69, 2000.

[14] S. Han, Y. Dang, S. Ge, D. Zhang, T. Xie, "Performance debugging in the large via mining millions of stack traces". In *Proc. of ICSE' 12*, pp. 145-155, 2012.

[15] C. Huang, I. Cohen, J. Symons, T. Abdelzaher, "Achieving scalable automated diagnosis of distributed systems performance problems". In *Technical Report*, HP, 2006.

[16] J. N. Hoover: "Outages force cloud computing users to rethink tactics". In *InformationWeek*, Aug. 16, 2008.

[17] J. Li, H. Shen, R. W. Topor, "Mining optimal class association rule set". In *Proc. of PAKDD' 01*, pp. 364-375, 2001.

[18] C. Liu, X.F. Yan, L. Fei, J.W. Han, S.P. Midkiff. "SOBER: statistical model-based bug localization". In *Proc. of ESEC/FSE' 05*, pp. 286-295, 2005.

[19] D. A. Patterson. "A simple way to estimate the cost of downtime". In *Proc. of LISA' 02*, pp. 185-188, 2002

[20] C. Sun, D. Lo, X.Y. Wang, J. Jiang, S.C. Khoo. "A discriminative model approach for accurate duplicate bug report retrieval". In *Proc. of ICSE' 10*, pp. 45-54, 2010.

[21] X.Y. Wang, L. Zhang, T. Xie, J. Anvik, J.S. Sun. "An approach to detecting duplicate bug reports using natural language and execution information". In *Proc. of ICSE' 08*, pp. 461-470, 2008.

[22] S. K. M. Wong, W. Ziarko, P. C. N. Wong, "Generalized vector spaces model in information retrieval". In *Proc. of ACM SIGIR' 85*, pp. 18-25, 1985.

[23] C. Yuan, N. Lao, J.R. Wen, J. Li, Z. Zhang, Y.M. Wang, W. Y. Ma. "Automated known problem diagnosis with event traces". In *Proc. of EuroSys' 06*, pp. 375-388, 2006.

[24] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, A. Fox, "Ensembles of models for automated diagnosis of system performance problems". In *Technical Report*, HP, 2005.

[25] D. Zhang, Y. Dang, J. Lou, S. Han, H. Zhang, T. Xie. "Software analytics as a learning case in practice: approaches and experiences". In *Proc. of MALETS' 11*, pp. 55-58, 2011.

[26] D. Zhang, S. Han, Y. Dang, J. Lou, H. Zhang, T. Xie. "Software Analytics in Practice". *IEEE Software*, Special Issue on the Many Faces of Software Analytics, vol. 30 no. 5, pp. 30-37, September/October 2013.

[27] D. Zhang, T. Xie. "Software analytics in practice: mini tutorial". In *Proc. of ICSE' 12*, pp. 997, 2012.