Autom Softw Eng (2017) 24:905–941
DOI 10.1007/s10515-017-0218-1

Experience report on applying software analytics in incident management of online service

Jian-Guang Lou[1] · **Qingwei Lin**[1] · **Rui Ding**[1] · **Qiang Fu**[2] · **Dongmei Zhang**[1] · **Tao Xie**[3]

Received: 1 December 2014 / Accepted: 12 June 2017 / Published online: 1 July 2017

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 2-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

✉ Jian-Guang Lou
jlou@microsoft.com

Qingwei Lin
qlin@microsoft.com

Rui Ding
juding@microsoft.com

Qiang Fu
qifu@microsoft.com

Dongmei Zhang
dongmeiz@microsoft.com

Tao Xie
taoxie@illinois.edu

1 Microsoft Research Asia, No.5 Danling Street, Beijing, People's Republic of China

2 One Microsoft Way, Redmond, WA, USA

3 Department of Computer Science, University of Illinois at Urbana-Champaign, 4237 Siebel Center, 201 N. Goodwin Ave., Urbana, IL 61801, USA

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.

Keywords Software analytics · Online service · Service incident diagnosis · Incident management

1 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 1 h service downtime for Amazon.com is $180,000 (Patterson 2002). Online services such as Amazon, Google, and Citrix have experienced live-site outages during the past couple of years (Inf 2008; Hoover 2008).

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[1]: 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

[1] Incident management, http://en.wikipedia.org/wiki/Incident_management.

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 three main aspects. First, incident management requires the service provider to take actions immediately to resolve the incident, as the cost of each hours service downtime is high (Patterson 2002). Second, as mentioned above, temporary workaround solutions rather than root cause fixing are often taken in incident management in order to recover the service as soon as quickly. Third, 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 diagnose the incident. Therefore, analyzing collected telemetry data is the only way for OCEs to conduct incident diagnosis and management.

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[2]) 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 (Zhang et al. 2008, 2013; Zhang and Xie 2012) 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 of Microsoft Research Asia, e.g., performance debugging in the large (Han et al. 2012), clone detection (Dang et al. 2012).

During the past several years, we have continuously studied real problems in this area. We formulate incident management of an online service as a software-analytics problem (Zhang et al. 2008; Zhang and Xie 2012), 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

[2] Microsoft SCOM, http://www.microsoft.com/en-us/server-cloud/products/system-center-2012-r2/default.aspx.

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 2-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 2 years to conduct algorithm research, build the diagnosis system, and make the system indispensable for the service-engineering team of Service X. After that, we continued our algorithm research and put efforts to apply such technologies to several online services in Microsoft.

In this paper, we share our lessons learned in such a project of an industrial system which are valuable for us and for the broad community of software engineering to carry out successful technology transfer and adoption.

1.1 Content organization

This paper is an extension of our previous ASE conference paper (Lou et al. 2013). Besides the content we have reported in (Lou et al. 2013), this paper includes our latest extensions and thoughts about the system according to the feedback and requirements in real usage from product teams in Microsoft along time:

1. New data source. Customer support ticket (i.e., reports) is a new and important data source for service management, which contains feedback from real customers.
2. More algorithms.
   - In order to help OCEs quickly narrow down their investigation on customer reported issues, we designed a algorithm to automatically identify attribute combinations that are highly related to an emerging issue, and help with problem localization.
   - In our previous ASE paper, we only reported our issue retrieval technique based on transactional logs, which is very effective for failure diagnosis. After a certain stage of service operation, the service has become stable, and latency and performance becomes another important aspect in service management. Based on this, we moved our focus to request latency and performance analysis. We found that transactional logs is not effective for latency issues. Then, we designed a new HMRF algorithm based on system metrics to facilitate the diagnosis of latency issues.

  - At the same time, for latency analysis, OCEs often want to understand the overview of all system metrics (e.g., CPU utilization) changing over time. In order to meet such requirement, we designed a fast time-series clustering algorithm and an intuitive UI to enrich the functionality.
3. New observation and experience from practices including a careful study on the incidents that our tool cannot help a lot, thoughts on technologies and data quality.

The rest of the paper is organized as follows. Section 2 introduces Service X. Section 3 presents our formulation of service-incident management as a software-analytics problem. Section 4 presents the resulting SAS system and its techniques. Section 5 discusses related work. Section 6 presents the lessons learned. Section 7 discussed some observations and implications we obtained from the project and Sect. 7 concludes the paper.

2 Background introduction: 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.

2.1 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.

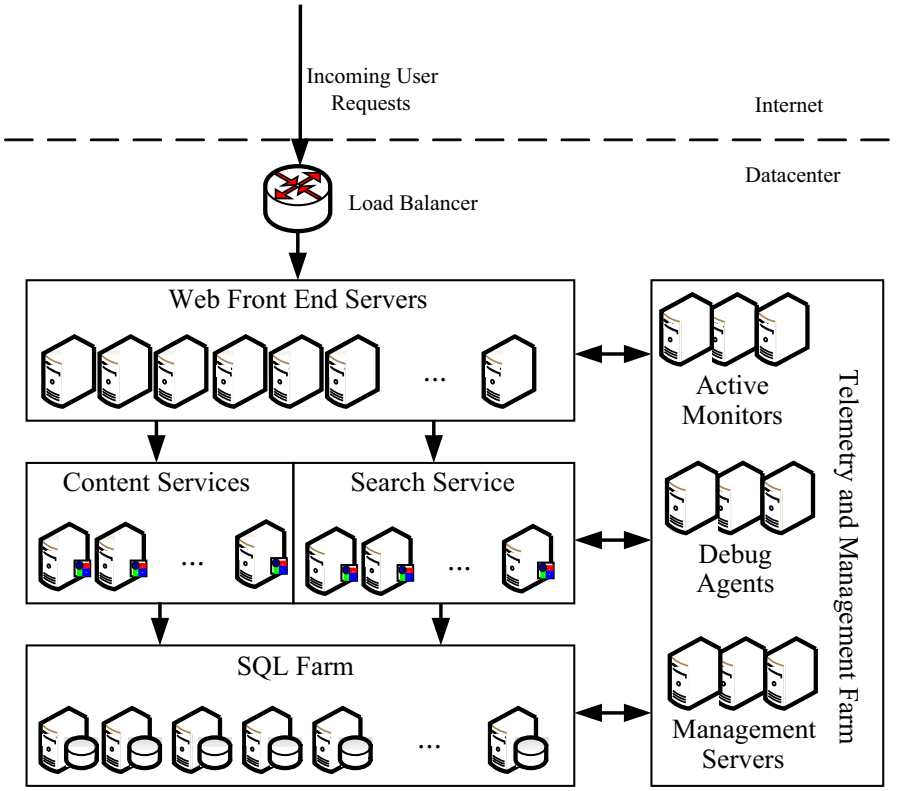

Fig. 1 System overview of Service X

2.2 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 incorrect 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 h 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.

3 Incident management as software analytics

As mentioned in Sect. 1, incident management of an online service by nature relies on analysis of various telemetry data collected from the service system, where 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. In our project, we formulate the incident-management problem as a software-analytics problem. We utilized the four-step approach of developing software analytics projects (Zhang et al. 2008; Zhang and Xie 2012) 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 SAS, we try to address the set of practical challenges in the incident management of Service X.

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.

3.1 Design goal

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 (Bodik et al. 2010; Cohen et al. 2004, 2005; Zhang et al. 2005) 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.

3.2 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.

Key Performance Indicators (KPI) Detecting incidents during service operation is often based on the KPIs, 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., $duration > 10$ s 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 min 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 KPIs 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.

System metrics 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.

Transactional logs 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 systems 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.

Customer support tickets Besides the data source we mentioned in our previous ASE conference paper (Lou et al. 2013), customer support reports were introduced to enrich the analysis of the tool during the past years. Whenever customers encounter an issue, they may report the issue to us and ask for supports. An issue can be caused by various reasons including software, hardware, and configuration problems at service side, client side, or the networking path from client to service. Despite immense efforts spent on software quality assurance, various types of issues still may occur in service systems in actual operations. As a large scale online service provider, we receive a large number of issue reports from our customers over a period of time. A typical issue report consists of many categorical attributes, such as *TenantType, ProductFeature, ProductVersion, SubscriptionPackage, DataCenter, Country, UserAgent, ClientOS*, and so on. Each attribute has associated values. An issue report also includes a time stamp recording the time at which the report was received. Therefore, the issue report data can be treated as multi-dimensional, time series data. It is also a responsibility of OCEs (along with customer support engineers) to address the problems from our customers and improve the experience of customers.

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.

4 Technical detail

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. After our first ASE conference paper(Lou et al. 2013), we further enriched our techniques (e.g., in Sects. 4.4 and 4.6) to handle new data sources and meet new requirements. 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 (Ding et al. 2012; Fu et al. 2012), we designed and extended our techniques based on carefully considering domain-specific characteristics of software-generated data to achieve satisfying performance.

4.1 Identification of performance beacon

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 intermediary 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) (Li et al. 2001). 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 ser-vice-incident beacons to the OCEs. The technical details of our analysis technique can be found elsewhere (Fu et al. 2012).

We tested some state-of-the-art algorithms proposed to solve similar problems (Bodik et al. 2010; Cohen et al. 2004, 2005), 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 h. Each incident contains only a small number of epochs. When a model is learned for each incident, the previously proposed learning algorithms (Bodik et al. 2010; Cohen et al. 2004, 2005) 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 (Bodik et al. 2010; Cohen et al. 2004, 2005) 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 (Fu et al. 2012) in the underlying incidents. The profound differences between classification and association-rule mining (Freitas 2000) can help illustrate why a mining-based technique works for Service X.

4.2 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 (Ding et al. 2012) . 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)[3] 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 $\triangle E_s = Int(c) \setminus Int(p)$ (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 $\triangle E_s$ are stored as candidates of suspicious execution patterns for further evaluation.

[3] Formal Concept Analysis, http://en.wikipedia.org/wiki/Formal_concept_analysis.

Second, we use a score named as Delta Mutual Information (DMI) to measure the suspicious level of each execution pattern. DMI is defined as $DMI(\triangle E_s) = M(X_c, Y) - M(X_p, Y)$, 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 $\triangle E_s$ for failure correlation (Ding et al. 2012). By walking through all edges in the lattice graph, we select all $\triangle E_s$ 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.

4.3 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 Sect. 2, 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 (e.g., an estimation of a median value is much more robust than an average):

$$\begin{cases} \mu = median(V) \\ \sigma = 1.48 \times median(|v - \mu|, \forall v \in V) \end{cases} \tag{1}$$

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.

4.4 Problem localization with multi-dimension analysis

As mentioned in Sect. 3, we added customer issue reports as a new data source for our analysis. Customers' issue report data is multi-dimensional, time series data. Most of the time, the team receives a relatively stable number of issue reports every day. As mentioned in Sect. 1, sometime the number of issue reports could suddenly increase. Such a burst means that there is a common issue encountered by a segment of user population due to major feature changes, mis-configuration, incompatible clients, environmental incidents, or software bugs. For example, Fig. 2 shows an emerging issue detected in 2013. Before December 8, 2013, the support team received an average of 70 issue reports per day. Starting from December 8, the volume of issue reports rose to over 300 per day. Among the issue reports received on December 8 and onwards, many share the following attribute combination: *Country="India"; TenantType="Edu"; DataCenter="DC6"*. We call these issues as "*emerging issues*". The emerging issues have negative impact on a large number of users, therefore, they should be addressed with a high priority.

In order to detect emerging issues, OCEs need to identify a particular attribute combination that can characterize the issues. The attribute combination can help isolate reports of an issue from other reports, provide hint for finding the root cause of the issue. For example, given the effective combination in Fig. 2, support engineers can quickly find out that these issue reports were related to a software configuration error, which fails to create accounts for customers in India, who are subscribed to the service through the EDU package. Therefore, many customers contacted the Microsoft support team asking for help. We call such an attribute combination "*effective combination*" since it characterizes an emerging issue. The effective combination associated with the emerging issue provided useful information for support engineers to isolate the problem and locate the potential cause of the issue.

We designed an algorithm to automatically and efficiently identify the effective attribute combinations that can characterize emerging issues for OCEs. Without this technique, OCEs have to identify potential effective combinations by manually exploring different attribute combinations with pivoting tools. The major challenge of effective-combination identification is its huge search space. If there are many attributes and each attribute has many different values, there will be an explosive number of possible attribute combinations, which makes the identification process very expensive or even impossible. In addition, the effective attribute combinations may be missed during manual exploration, since the coverage of manual exploration cannot be very high. Furthermore, detecting emerging issues could be very difficult, because the burst of an emerging issue can be easily lost within the background noisy issue reports and no clear burst can be observed in the overall trend of all reports.

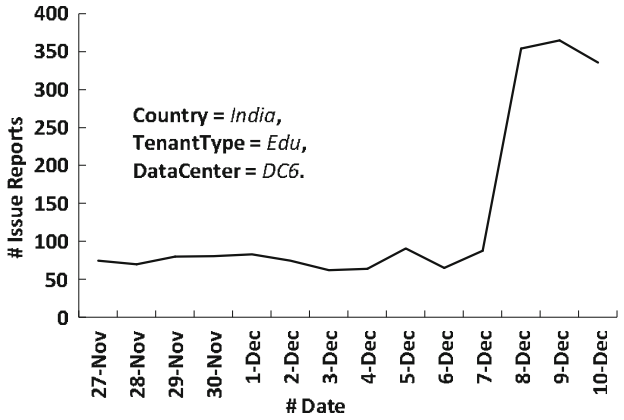

Fig. 2 An example of emerging issue

The core algorithm of our technique is to effectively reduce the search space without missing the effective combinations. In order to achieve this goal, we have designed the following three search space pruning strategies.

- *Impact based pruning* Each effective combination should be related to a large volume of issue reports, which means that it has impacted a large number of customers. We adopt an impact-based strategy to prune the attribute combinations associated with small numbers of issue reports. Here, the *impact* of an attribute-combination is defined as the number of its related issue reports.
- *Spike detection based pruning* Effective combinations should be able to reflect significant volume increases of issue reports. Therefore, in the search process, we prune off the attribute combinations that exhibit small or no spike in issue-report volume. The underline intuition is that a real serious issue always causes a bust of issue-report volume.
- *Isolation power based pruning* An effective combination is the node that can exactly split the entire dataset into two parts: with and without a significant spike of report volume. In other words, an effective combination should be able to isolate the attribute combinations that exhibit a volume spike from the other combinations that do not. We propose the notion of *Isolation Power* to measure such kind of isolating capability for an attribute combination. During the search process, if the current set has a higher isolation power than its direct supersets and subsets, then the current set is an effective attribute combination. In this case, all its subsets will not be searched. We use this strategy to remove the possible redundancy in the identified effective combinations, and to make the results concise and compact. The detail of *Isolation Power* can be found in our other paper (Lin et al. 2016).

With our technique, OCEs can quickly detect an emerging problem that may influence the experience of a large number of customers based on the reports. At the same time, the discovered attribute values in an effective combination can provide rich information for OCEs to locate the source of the problem. As an extension of SAS, we have successfully applied this technique to Service X in production.

4.5 Leveraging previous effort for recurrent incidents

Similar incidents may reoccur due to reasons discussed in Sect. 3.1. 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.

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. As we have mentioned in Sect. 3, there are mainly two kinds of monitoring data, which are system metrics and system logs.

In practice, we found that system logs are often more suitable for troubleshooting availability issues (e.g., an unplanned interruption of the service). When an availability issue happens, engineers usually inspect the transactional logs to determine whether there is any clue of the underlying problem. On the other hand, system metrics are often more useful on investigating latency issues (e.g., unplanned degradations of average response time in the service). The reason is that, each component of the system has collected a set of performance counters including latency, work load, resource usage, and these performance counters often contain sufficient information for latency-issue investigation. Therefore, for availability incidents, we use log as the source information for incident retrieval, and for latency incident, we use performance counters as the source information for incident retrieval. It is our future work to correlate and integrate the suspicious signatures and beacons to determine root cause more accurately.

Transactional-log based retrieval

For incidents related to system availability drop, we mainly use the transactional log as the source information for analysis. Using the technique discussed in Sect. 4, 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 in repository. We treat each incident as a document, each execution pattern as a term, and the corresponding DMI score as the weight of the term. Let $D = \{d_1, d_2, \ldots, d_m\}$ be the total m documents in the issue repository. Consider the given new issue as a query, denoted as q. So we represent each document d_i as $\sum_{p \in A(i)} w_{ip}\mathbf{t_p}$, where a term is represented by an abstract vector $\mathbf{t_p}$, p is the index of the corresponding term in d_i, $A(i)$ is the valid index set, and w_{ip} is the weight of $\mathbf{t_p}$. We use DMI as the weight for each term, because suspicious execution patterns are often used to distinguish between different incidents in the practice of software engineering. Such weight is much different from the *TF-IDF* weight (Han et al. 2011). Our evaluations proved that it is better than *TF-IDF*. We then use the Generalized Vector Space Model (Wong et al. 1985) to calculate the similarity of two incidents. We calculate the cosine score of two document vectors, each representing one issue. In particular, given a documents d_i that $d_i = \sum_{p \in A(i)} w_{ip}\mathbf{t_p}$. We next define the similarity metric

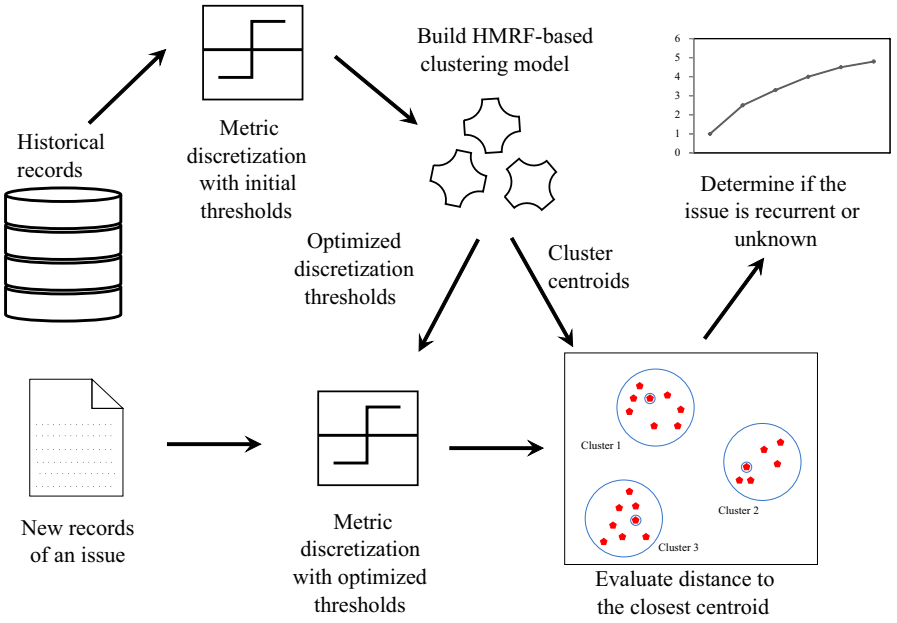

Fig. 3 Overview of HMRF-based incident association

$$sim(d_i, d_j) = \frac{d_i \cdot d_j}{\|d_i\| \, \|d_j\|} = \frac{\sum_{p \in A(i), q \in A(j)} w_{ip} w_{jq} \mathbf{t_p} \cdot \mathbf{t_q}}{\|d_i\| \, \|d_j\|} \tag{2}$$

The metric measures the cosine of the angle between the two vectors. Here $\|d_i\| = \sqrt{d_i \cdot d_j}$. We define inner product between two terms:

$$\mathbf{t_p} \cdot \mathbf{t_q} = \#\text{ of overlapped events in } p\text{-th term and } q\text{-th term} \tag{3}$$

System Metric based retrieval

For latency incidents, we proposed a new algorithm to improve the effectiveness of recurrent issue identification based on system metrics. In our approach, the metric data is treated as observations and the corresponding issue types are treated as a field of hidden variables. As described in Fig. 3, given the historical data of performance issues, we first discretize the metric data according to a set of initial thresholds. We then build a HMRF-based model that can cluster issues according to the issue types, and derive optimal thresholds and clustering parameters using an EM (Expectation-Maximization)-based algorithm [refer to Figure 4 of our paper (Lim et al. 2014)]. The SLO compliance records are treated as the non-issue type. Finally, when a new record arrives, we determine if it indicates a recurrent issue or a brand-new issue by evaluating its distance to the centroid of clusters.

In our scenario, we try to understand the underlying run-time performance states of the system through analyzing the observed values of system metrics. In practice, it is customary to expect a strong time-based correlation among the values of

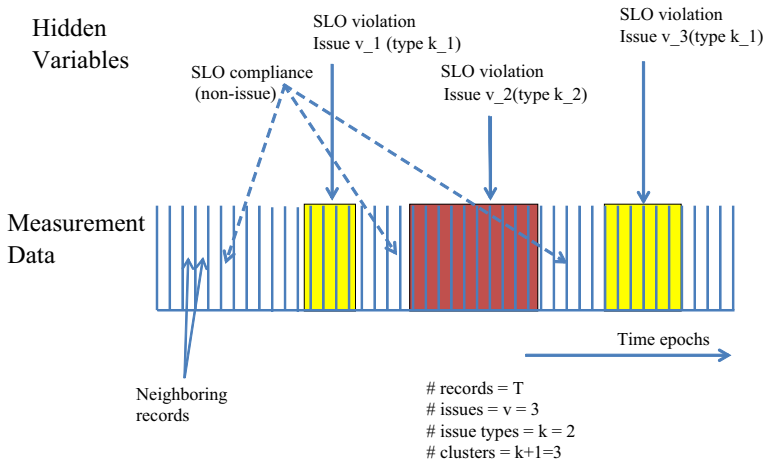

Fig. 4 An illustrative example of our HMRF model

system metrics within each service incident (Bodik et al. 2010; Duan and Babu 2008). The metric values of two neighboring time-epochs often reflect similar performance state. HMRF is a good tool to model the relationship among neighboring epochs. It has been successfully used in many areas, including semi-supervised clustering (Basu et al. 2004), bioinformatics (Li et al. 2009), because they support probabilistic modeling of information about the dependencies between hidden variables and their observations, and the mutual influences among neighboring observations. In this project, we define a temporal-neighboring constraint, which means that every pair of neighboring time-epochs tends to be of the same incident. Suppose that there exist v instances of performance issue and each of which belongs to one of the k issue types ($k \leq v$). The v issue instances and the non-issue instances (i.e., compliance instances) can be grouped into $k+1$ clusters (k issue types plus 1 non-issue type). Each cluster consists of performance issues of the same type. An example of issue clustering is shown in Fig. 4. There are three issue instances, which consist of records of two issue types (i.e., k_1, k_2). The SLO compliance records can be treated as a special non-issue type. The problem is to automatically group the T records into 3 clusters, and assign a new record to a correct cluster. If the new record can be well categorized into a cluster, a reoccurring issue is detected. Otherwise, the issue is a brand-new issue. The detail implementation of this algorithm can be found from our technical report (Lim et al. 2014).

4.5.1 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

Table 1 Healing actions

Verb	Target	Event of location
Reboot	SQL (Database)	ev1
Recycle	App-Pool (Application Pool)	ev2
Restart	IIS (Internet Information Service)	ev2
Re-image	WFE (Web Front End)	ev2
Reboot	WAP (Web Application)	ev3
Patch	Service (SQL/WFE/WAP)	ev1, ev2, ev3
Restart	Scanner (Anti-virus Component)	ev2, ev3
Restart	Search (Search Component)	ev3
Restart	AD (Active Directory)	ev4

"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 the SQL machine SQL32-003". We determine the location using the technique that detects the malfunctioned server role.

4.6 Fast time series clustering

In order to ensure service quality, various types of performance counters (e.g., CPU usage, disk I/O, network throughput, etc.) are continuously collected on each server. For analysis purpose, they are often aggregated at pre-defined time intervals (e.g., 5 min) on each server, resulting in time series representing certain performance characteristic of the service(s) under monitoring. When conducting incident management, OCEs often need to explore these performance counters for understanding data characteristics, spotting patterns, and validating hypotheses, etc. by leveraging different analysis techniques such as clustering, matching, filtering, and visualization, etc. These tasks help users obtain insights and make informed decisions. The process of completing these tasks is usually iterative, which requires analysis techniques to be fast and scalable in order to create real-time and interactive exploration experiences.

Clustering time series is an important and useful technique for exploratory study on the characteristics of various groups in a given time series dataset. It can identify the homogeneous groups of time series data based on the similarity among different time-series. However, given the scale of our Service X, it is challenging to cluster a huge number of time-series to support interactive time-series exploration. In order to overcome this problem, we proposed a fast time-series clustering algorithm with the aid of an elegant sampling mechanism [the algorithm details can be found in (Ding et al. 2015)].

Based on the fast clustering algorithm, we designed a friendly UI (as shown in Fig. 5) for users to explore time-series data, where OCEs can get an overview of all time-series groups by clicking the Groups on the left panel. In the figure, a time

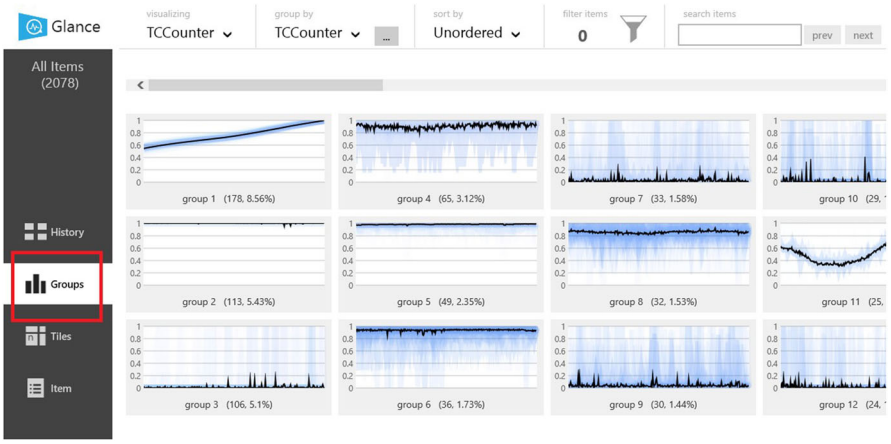

Fig. 5 Interactive time-series exploration

series dataset including data of CPU utilization for 2078 servers, is presented. Each time series instance has 1,008 data points. The analysis task is to obtain an overall understanding on the CPU utilization across all the servers, and find out how these two aspects relate to each other. With our fast clustering technique, 21 groups of CPU utilization are immediately obtained in less than 2 s. Each group in fact is a cluster obtained from our clustering algorithm. In each chart, the black curve is the median of the group, (i.e. median is used to aggregate all the time series instances in the group). The above groups reveal different patterns of CPU utilization of different servers. For example, the chart shows that we have increasing CPU usage for 8.56 % machines. OCEs can further look into the individual time-series items of a group by clicking the chart of the group. Our high-performance clustering algorithm enables real-time and interactive drill-down analysis in our tool. With such an interactive exploration, OCEs can quickly understand the overall resource usage information (e.g., CPU usage) of the system, and discover some suspicious resource-usage patterns immediately.

4.7 Evaluation

4.7.1 Performance beacon

We evaluated our technique of performance beacon analysis 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 ($\sim$ 90%) compared to the recall of $\sim$ 60% obtained using L1-Logistic Regression [in short as L1-LR, an algorithm in state-of-the-art research (Cohen et al. 2005)]. Figures 6 and 7 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

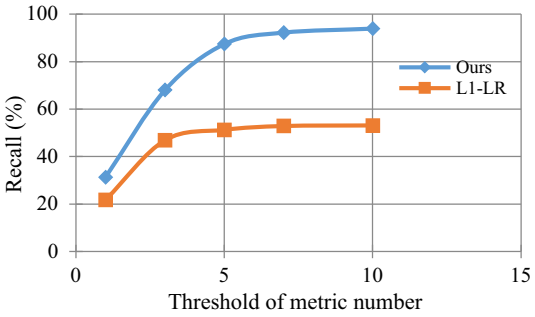

Fig. 6 Recall of beacon analysis

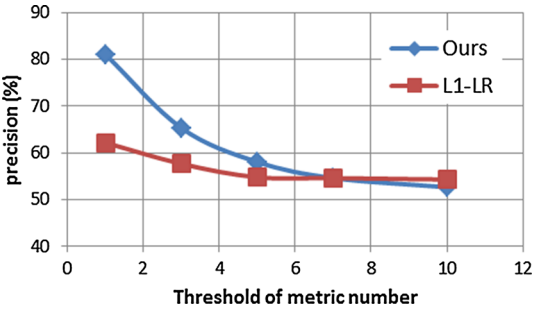

Fig. 7 Precision of beacon analysis

and coupling effect, are common among many other online services. Therefore, our analysis technique can also be applied to other online services.

4.7.2 Multi-dimension analysis

We use *F-measure*, *Recall*, and *Precision* metrics to measure the effectiveness of our algorithm for fault localization based on multi-dimension analysis. The *F-measure* is defined as follows:

$$F\text{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

where $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$. Here, TP (true positive) is the number of actual effective combinations correctly reported by our algorithm. FP (false positive) is the number of wrongly reported effective combinations. TN (true negative) is the number of non-effective combinations that are correctly reported. FN (false negative) is the number of effective combinations that are not reported by our algorithm. The higher the metric values, the better the detection performance. In addition, we measure the execution time (in seconds) to evaluate the efficiency of the algorithm.

Table 2 Evaluation results of fault localization with multi-dimension analysis

Metrics	Our technique	DPMiner
Recall	0.84	0.83
Precision	0.88	0.37
F-meaure	0.86	0.51

Table 3 Healing actions in studied issues

Category ID	Verb	Target	# of cases
ID1	Recycle	App-Pool	57
ID2	Reboot	WAC	57
ID3	Restart	IIS	43
ID4	Reboot	SQL	39
ID5	Restart	AD	31
ID6	Re-image	WFE	9
ID7	Patch	Service	3
ID8	Restart	Scanner	2
ID9	Restart	Search	2

We collected the actual issue report data, which contains more than ten thousand issue reports collected in early 2015. There are total 92 emerging issues in this dataset, which are verified by the domain experts in the product team.[4]

To the best of our knowledge, there is not specific algorithm in literatures for similar scenario. Therefore, we are not able to find a good basis for comparison study. In this paper, we only compared our technique with a classical algorithm (DPMiner). DPMiner (Dong and Li 1999) detects emerging patterns, which are attribute-sets whose support rates increase significantly from one dataset to the other. As shown in Table 2, our technique is able to achieve good results. The Recall, Precision, and F-measure values are 0.84, 0.88, and 0.86, respectively. Furthermore, it significantly outperforms the DPMiner approach. The improvement on F-measure is 69%. Note that emerging pattern mining techniques such as DPMiner is not originally designed for mining effective combinations defined in our scenario because it does not support change detection in time series data. Also, there are no pruning strategies based on isolation power.

4.7.3 Heal suggestions

We also evaluated the effectiveness of our techniques for two different types of incidents (i.e., availability incidents and latency incidents) respectively.

Transactional-log based retrieval

[4] We choose this dataset because it has quality labels verified by domain experts. Although we do have other real datasets that have a larger number of attributes, the quality of their labels may be insufficient to calculate TN and FN due to the difficulties of manual identification effort mentioned above.

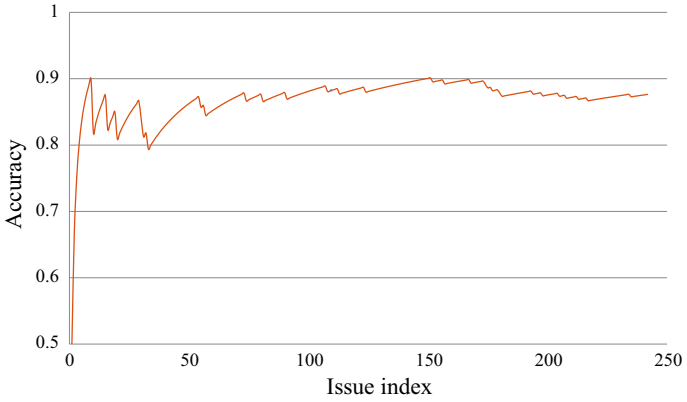

Fig. 8 Accuracy of suggesting correct healing action

We evaluate our technique of availability-issue retrieval in Service X. We studied 243 service issues (they were detected by our internal monitoring system) in the year of 2012. Each of them has a clear resolution and its related transaction logs. The healing actions for the 243 issues are categorized into 9 categories based on the combination of their verb and target information as shown in Table 3. To comprehensively evaluate our approach, we design experiment to replay the major real usage of our technique. Specifically, we in-turn treat each of the 243 issues (in the order of their occurring time) as a "new issue", and then we reflect real usage of our approach in practice by treating the previously encountered issues (i.e., those that occurred before "new issue") as the "historical issues". We then apply our approach for each combination of "new issue" + "historical issues" and then measure the accuracy of our approachs effectiveness in suggesting a correct healing action for the "new issue".

Figure 8 shows the overall accuracy trend for each approach. The X-axis is the index of each issue (sorted by occurring time); the Y-axis is the average accuracy of the issues between the first one and the current one. Higher accuracy values indicate better effectiveness. According to Fig. 8, the overall accuracy of our approach is 87%, specifically, our approach correctly suggests healing actions for 213 issues. The high accuracy of our approach is critical to enable auto-healing tasks. Although currently service recovery heavily relies on manual efforts, product teams are starting to deploy some scripts to apply healing actions automatically, e.g., deploying a script in a dedicated management machine to command the IIS of a remote service to restart. We can then map our suggested healing action to its corresponding script, which is deployed to accomplish service auto-healing. More detailed results including the ROC curves of our technique can be found in (Ding et al. 2012, 2014).

Metric based retrieval

Similar to the latency issue retrieval, we also extensively evaluated our HMRF-based approach in the practice of Service X. Figure 9 shows the performance of HMRF-based approach in detecting recurrent issues of Service X, measured using ROC curves. The TPR (*True Positive Rate*) value is around 98% when FPR (*False*

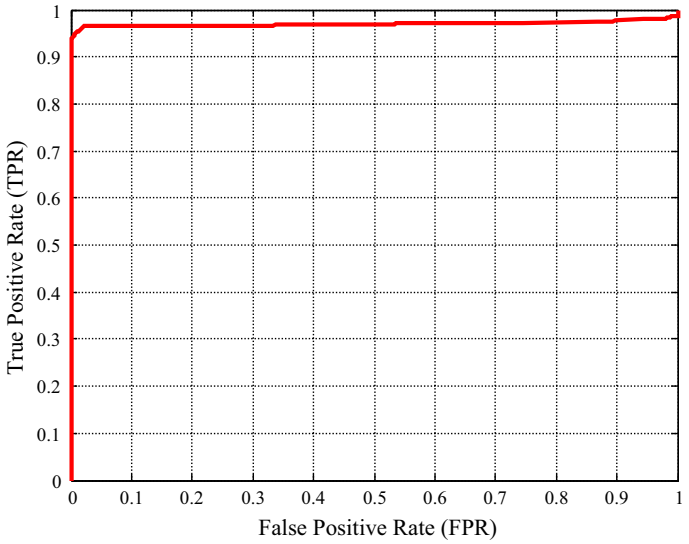

Fig. 9 ROC curves of the proposed approach for recognizing recurrent performance issues

Positive Rate) value is 10%, and the AUC (*the area under the ROC curve*) value is 96.91%. These results are fairly good for a healing suggestion system. More detailed results including the comparison between our technique and other two state-of-art algorithms can be found in (Lim et al. 2014).

5 The tool

5.1 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.

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 Fig. 10, 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

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 SQLServers	No malfunctioned SQLservers 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

Fig. 10 An example analysis report

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 Fig. 10, 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.

5.2 Deployment

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. 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 value of SAS, 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 year study, about 90% 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 usage data shows that SAS helped diagnose about 76% of the service incidents that SAS was used for.

To make our approach more effective and better fit into pipelines of service diagnostic in real production, we investigate and record the issues or conditions where our approach fails to help. There are following 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. Another typical case is that the issue is caused due to network devices before the front-end server of Service X, so the user requests did not reach the front-end service at all.
- Second, some detected incidents are false alarms. Typical examples are the issues detected during system upgrading, i.e., the monitoring system was not shut down in time when the service upgrade began. Including such kind of data in the system can hurt the accuracy of the tool.
- Third, some incidents are "one-shot issue". Each incident is unique and is not similar to any other ones. Our approach fails on these issues because there exist no similar historical issues for these issues. According to the feedback on these issues from engineers, the performance beacons and suspicious log patterns still can provide useful information for diagnosis, and the signatures that our approach generates are still valuable to the engineers in diagnosis.
- Fourth, sometime, the telemetry data is insufficient for diagnosis. We have some cases, where even knowledgeable developers cannot identify the causes by inspecting the telemetry data. More information should be recorded in log messages for generating more proper signatures. For example, from the systems source code, there is one event generated for indicating the overall general exception handling at the last stage of request processing. If the request fails, there could be various

exception types. However, if there was no exception call-stacks, it is often quite difficult to know where the exception is thrown.

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.

6 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 the following categories.

Incident-beacon identification Previous work (Bodik et al. 2010; Cohen et al. 2005, 2004; Huang et al. 2006; Zhang et al. 2005) 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. (2004) and Huang et al. (2006) 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. Zhang et al. (2005) extended Cohen's work (2004) to changing workloads and external disturbances adaptation via maintaining an ensemble of TAN models, such that a new model is added whenever existing ones do not accurately capture the current system behaviors. Cohen et al. (2005) extended their previous approach (Cohen et al. 2004) by proposing a Signature approach. Bodik et al. (2010) 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 (Bodik et al. 2010; Cohen et al. 2004, 2005; Huang et al. 2006; Zhang et al. 2005) 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.) (Fu et al. 2012). 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. Some techniques analyze performance metric data, and apply machine learning techniques to classify issues. For example, Falcon system (Duan and Babu 2008) utilizes both labeled and unlabeled data to improve the overall accuracy of diagnosis. It is an active-learning technique, which facilitates manual labeling effort via maximizing the benefits gained from newly-diagnosed unknown instances. Natu et al. (2011) focused on automated debugging of SLO violations by initiating a two-stage process of feature selection using a Classification and Regression Trees (CART) method followed by a statistical change-point detection algorithm. Yuan et al. (2006) 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 (Bodik et al. 2010; Cohen et al. 2004, 2005; Huang et al. 2006; Zhang et al. 2005) 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. (2009) 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 (Sun et al. 2010; Wang et al. 2008) 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 (Ding et al. 2012). 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 (Liu et al. 2005; Sun et al. 2010) in that we all leverage the differences between the logs of failed and succeeded requests. Sun et al. (2010) evaluated patterns mined from both correct and incorrect runs to detect duplicate bug reports. Sambasivan et al. (2011) have also developed a technique for performance problem localization by comparing request flows from two executions (e.g., of two system versions or time periods). The request flows are built based on end-to-end request-flow tracing within and across service components. Nagaraj et al. (2012) proposed DISTALYZER, an automated tool to support developer investigation of performance issues in distributed systems by identifying salient differences between sets of logs that may potentially affect overall performance and significantly contribute to the observed differences in behavior. Similar to these approaches, our work uses contrast information to achieve high accuracy of signature generation. Cellier (2008) 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.

There are another set of related literatures that are loosely related to our fault site localization technique based on multi-dimension analysis. Li et al. (2011) studied 18 software usage characteristics and investigated how they are related to field quality and how they differ between pre- and post-release. They identified the five most important usage characteristics through general linear regression. Menzies et al. (2013) proposed machine-learning based methods for learning from bug datasets succinct rules that explain quality issues. Bird et al. (2014) found that the reliability of a software system depends on the environment it operates in. They performed an empirical study of more than 200,000 Windows users, and found that the reliability of individual systems is related to whether and which other systems are installed. They also applied

association rule mining to detect the influence of factor combinations on the reliability of a system. Epifani et al. (2010) addressed the problem of identifying change points concerning the reliability and performance of a software service. Their approach is based on the execution trace produced by client invocations, and only tries to find out change points. Our multi-dimension algorithm works with the customer issue reports. It identifies not only the change point, but also the effective attribute combinations that are associated with the changes. In our algorithm, we integrate closed itemset mining, change detection, and pruning techniques to detect emerging issues in multi-dimensional time series issue data.

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.

7 Lessons learned

We started the project on data-driven performance analysis for online services in 2010. It took us 2 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.

7.1 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 Sect. 4.1; 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.

7.2 Improving techniques in practice

Robustness The built service should provide users with robust models and analysis results, otherwise users are not willing to spent their critical time on it. We improve the robustness of the tool from both the data level and the algorithm level.

On the data level, strict data quality check is performed before the data is used further analysis. In a large-scale online-service system, data missing and noise can be common. In order to obtain high quality system models, we need clean data. In our tool, we monitor the data quality. For example, the number of data missing and noise in a training data set are required to be less than 2% of total volume of data. Otherwise, the data cannot be used to train a model. The model is trained at a fixed frequency using up-to-date data so that the created rules always consider the latest data pattern information.

On the algorithm level, robust and mature algorithms have been adopted by both components. 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 subset 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 (2008)].

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 min, collects the metric data newly generated during the past 5 min, 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 Sect. 4.1) 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-min 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.

7.3 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 engineer by mistake.

7.4 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 trouble-shooting, 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 different scenarios where SAS was used for or not used for certain incidents could let us to find new research opportunities.

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.

8 Observations and discussion

8.1 Practical implications

Before the discussion, let's look at a real case first, as this case demonstrates the capability and limitation of our data-driven technique in the area of incident management.

The case is about Antivirus Configuration Corruption. In January of 2012, ServiceX experienced continuous performance problem in one datacenter. During the occurrence of this issue, customers experienced both slow response and failures of uploading files in an unpredictable fashion. Engineers who first diagnosed this issue found that one Web Front End (WFE), named WFEx, "produced" most *http-status* = 500 errors. During the investigation, SAS automatically locates the problem to WFEx, and finds the symptoms of the issue including "Internal Server Error" messages and "SQL failing over detected" messages. Considering that the load-balancing strategy randomly selected one WFE to serve each transaction instance, and the transaction instances that go through WFEx have a high probability to fail, the engineers asserted that WFEx went into a bad state. They try to recover the service by rebooting WFEx. However, the rebooting action did not turn back the service availability, and the same issue existed continuously. They have to rotate out WFEx for off-line investigation. After several senior experts investigated the problem for a long time, they finally found the root cause of the issue is "*configuration file for antivirus component became corrupted after a random restart*" (denoted as ACC). Resolving this issue is challenging because the antivirus component is a third-party component, and the team do not collect enough telemetry data about it. It was finally resolved in about two weeks by involving 12 experts in different relevant teams and investigating various logs of different components/features. Similar to other issues, SAS automatically recorded this issue in the repository. On early February, 2012, a new issue X occurred in another farm of the same datacenter, our approach retrieved the historical issue of ACC as the most similar issue, with the similarity score of 0.96. Guided by the information of the historical issue and its healing suggestion, the engineer, who was not familiar with this issue, immediately moved to check the antivirus configurations instead of rebooting the WFE (a common healing action) and successfully recover the WFE in a few minutes. During this second occurrence, our approach helped reduce much investigation time of the engineer by providing informative diagnostic clues.

From the case, we can see that SAS can quickly locate the problem site and find out the most relevant log messages by screening a large amount of telemetry data. However, it does not help much when a specific case happens at first time. Its capability of correctly associating a recurrent issue to the past one did help engineers a lot when the issue happens again.

While SAS has been proven to be useful in Service X, we now discuss some of the practical implications of SAS.

At first, SAS mainly consists of a set of modules using statistical data analysis techniques. In order to make the analysis statistically meaningful, there must exist enough data to provide high coverage of system run-time behaviors. For example, in the above case, the tool cannot find deep information about the cause which are not recorded by the logs at all. The basic assumption here is that the collected logs would capture artifacts of the problem under investigation and hence point to that component.

Second, although SAS can help engineers quickly take actions to recover Service X for many cases. However, some issues still need intensive manual efforts during their first-time occurrences as described in the above example. The insights obtained from data-driven techniques are usually correlation rather than causation. There is still a gap between our analysis results and root causes. Root cause analysis often needs decent

knowledge about the service system and deep experience in software engineering. In SAS, we design the tool to support human-in-the-loop (HITL) for decision making and knowledge inputing. Our healing adaptation algorithm is another technique to leverage the knowledge in the issue repository. Studying on proper methodologies for representing complex domain knowledges and integrating them with data-driven techniques is still a very important part of our future work.

8.2 Improving telemetry

As a data-driven analysis project, data quality is a key problem here. In addition to inventing various techniques of system/service data analysis, we also found that we need to pay much attention to another fundamental problem: how to generate logs or system metrics to make sure that they contain precise and concise information necessary for incident management? In fact, the importance of logging can be widely identified by the various usages of logs in diverse software system management tasks and techniques, including anomaly detection (Fu et al. 2009; Lou et al. 2010), error debugging (Glerum et al. 2009), performance diagnosis (Nagaraj et al. 2012; Sambasivan et al. 2011), workload modeling (Sharma et al. 2011), system behavior understanding (Fu et al. 2013), etc. Based on our observation during the project, we summarize a number of directions that deserve further exploration for improving current logging practices, which in turn to benefit both the effectiveness and the efficiency of the incident management (Fu et al. 2014).

End-to-end tracing Modern software systems are generally composed of various components, which may be deployed as distributed systems. End-to-end tracing can provide a detailed picture of how a request was serviced through the whole system, and thus can assist in understanding the behaviors of a complex system. In SAS, we heavily depend on the logs generated by end-to-end transactional logs. Each request contains a unique request ID that can be used to correlate the log messages of a user request from different service components together to form a whole log sequence (see Sect. 3.2).

Effective logging With a system scaling up, the size of produced logs becomes large, e.g., at a rate of about 50 gigabytes (around 120–200 million lines) per hour. However, most of them are not relevant when we try to diagnose a service incident. Hence, finding useful information under the huge volume of log data is referred to colloquially as "finding the needles in the haystack". In SAS, we try to use analysis techniques to filter out irrelevant logs and system metrics. However, a more efficient way may be preventing the generation of irrelevant logs to eliminate the processing overhead caused by irrelevant logs, namely "On-demand logging in production". Traditionally, logging statements are statically inserted to source code, and print log messages at specific fixed program locations, which may generate too many useless logs. One promising direction is to log on demand. That is, each program location for logging can be dynamically enabled (or disabled) to generate logs when a specific condition is satisfied (or not satisfied). For example, at a logging point of a Remote Procedure Call (RPC), only latency above a threshold value is symptomatic for logging as a performance anomaly, whereas latencies of normal calls can be ignored.

Log categorization Logs should be categorized by source, type, and function. For example, in our system, each logging statement has a unique eventID (see Sect. 3.2). With event IDs, we can easily construct a rough code path from a log sequence. Such log categorization can help achieve better log understanding and efficient postmortem analysis. Otherwise, we need extra efforts to extract a signature for each type of log entry (Fu et al. 2009). One good example is the Unified Logging System (ULS) in Microsoft, which supports automatic tagging of logs, such as event ID and request ID. With these automatically recorded tags, logs can be easily categorized with respect to an event type or a request. However, more of such similar infrastructure features are needed. For example, since we have more and more cross workloads, it would be quite helpful for troubleshooting if logs can be correlated to different workloads (i.e., request type).

Closed-loop process for log quality Although all practitioners know that quality of logs is very important, however, limited efforts and attentions have been put to ensure the high quality of log data. During our project, we found that many difficulties are caused by the quality of logs, and there is still a big room for the improvement of log quality. For example, unlike Service X, many other systems do not record a request ID in log data, and do not have an event id for each log entry. Without request IDs, we cannot link log messages related to the operations of a single request together. This brings extra computational cost and accuracy degradation to our algorithms. In the state-of-art practice of software engineering, there is no strong incentive for engineers to put efforts and attentions on log quality. Many engineers write their logging statements with an ad hoc way. In the current practice, we do not have a process to ensure the quality of logs. It is totally different from that of source code, where we have a closed-loop process (i.e., develop, code review, testing, and fixing) to ensure code quality. Some teams do have guiding documents for logging practice. However, such logging guidances are often loosely followed. There is no formal process to enforce the logging practice, e.g., no review, no testing, and no feedback. In addition, different from source code, there is no existing good practice for engineers to test the quality of logs. Given the importance of logs in the cloud era, we argue that it is worthy to put efforts on researching and developing new technologies and processes for ensuring log quality, such as quality evaluation, testing, closed-loop process, and so on.

9 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, the resulting industrial system, and the lessons learned from the process of research development and technology transfer.

References

Amazons s3 cloud service turns into a puff of smoke. In: InformationWeek NewsFilter (2008)

Ashok, B., Joy, J.M., Liang, H., Rajamani, S.K., Srinivasa, G., Vangala, V.: Debugadvisor: a recommender system for debugging. In: Proceedings of ACM FSE'09, pp. 373–382 (2009)

Basu, S., Bilenko, M., Mooney, R.: A probabilistic framework for semi-supervised clustering. In: Proceedings of SIGKDD, pp. 59–68 (2004)

Bird, C., Ranganath, V.P., Zimmermann, T., Nagappan, N., Zeller, A.: Extrinsic influence factors in software reliability: a study of 200,000 windows machines. In: Proceedings of ICSE, pp. 205–214 (2014)

Bodik, P., Goldszmidt, M., Fox, A., Woodard, D.B., Andersen, H.: Fingerprinting the datacenter: automated classification of performance crises. In: Proceedings of EuroSys, pp. 111–124 (2010)

Cellier, P.: Formal concept analysis applied to fault localization. In: Proceedings of ICSE, pp. 991–994 (2008)

Cohen, I., Chase, J.S., Goldszmidt, M., Kelly, T., Symons, J.: Correlating instrumentation data to system states: a building block for automated diagnosis and control. In: Proceedings of OSDI, pp. 231–244 (2004)

Cohen, I., Zhang, S., Goldszmidt, M., Symons, J., Kelly, T., Fox, A.: Capturing, indexing, clustering, and retrieving system history. In: Proceedings of SOSP, pp. 105–118 (2005)

Dang, Y., Zhang, D., Ge, S., Chu, C., Qiu, Y., Xie, T.: Xiao: Tuning code clones at hands of engineers in practice. In: Proceedings of ACSAC, pp. 369–378 (2012)

Ding, R., Fu, Q., Lou, J.G., Lin, Q., Zhang, D., Shen, J., Xie, T.: Healing online service systems via mining historical issue repositories. In: Proceedings of ASE, pp. 318–321 (2012)

Ding, R., Fu, Q., Lou, J.G., Lin, Q., Zhang, D., Xie, T.: Mining historical issue repositories to heal large-scale online service systems. In: Proceedings of DSN, pp. 311–322 (2014)

Ding, R., Wang, Q., Dang, Y., Fu, Q., Zhang, H., Zhang, D.: Yading: Fastclustering of large-scale time series data. In: Proceedings of VLDB, ACM, pp. 473–484 (2015)

Dong, G., Li, J.: Efficient mining of emerging patterns: discovering trends and differences. In: Proceedings of SIGKDD, ACM, pp. 43–52 (1999)

Duan, S., Babu, S.: Guided problem diagnosis through active learning. In: Proceedings of ICAC, pp. 45–54 (2008)

Epifani, I., Ghezzi, C., Tamburrelli, G.: Change-point detection for black-box services. In: Proceedings of FSE, pp. 227–236 (2010)

Freitas, A.A.: Understanding the crucial differences between classification and discovery of association rules—a position paper. In: SIGKDD Exploration, vol. 2(1), pp. 65–69 (2000)

Fu, Q., Lou, J.G., Wang, Y., Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis. In: Proceedings of ICDM, pp. 149–158 (2009)

Fu, Q., Lou, J.G., Lin, Q., Ding, R., Zhang, D., Xie, T.: Performance issue diagnosis for online service systems. In: Proceedings of SRDS (2012)

Fu, Q., Lou, J.G., Lin, Q., Ding, R., Zhang, D., Xie, T.: Contextual analysis of program logs for understanding system behaviors. In: Proceedings of Mining Software Repository, pp. 397–400 (2013)

Fu, Q., Zhu, J., Hu, W., Lou, J.G., Ding, R., Lin, Q., Zhang, D., Xie, T.: Where do developers log? an empirical study on logging practices in industry. In: Proceedings of ICSE (2014)

Glerum, K., Kinshumann, K., Greenberg, S., Aul, G., Or-govan, V., Nichols, G., Grant, D., Loihle, G., Hunt, G.C.: Debugging in the large: ten years of implementation and experience. In: Proceedings of SOSP, pp. 106–116 (2009)

Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques, 3rd edn. Morgan Kaufmann, Burlington (2011)

Han, S., Dang, Y., Ge, S., Zhang, D., Xie, T.: Performance debugging in the large via mining millions of stack traces. In: Proceedings of ICSE, pp. 145–155 (2012)

Hoover, J.N.: Outages force cloud computing users to rethink tactics. In: InformationWeek (2008)

Huang, C., Cohen, I., Symons, J., Abdelzaher, T.: Achieving scalable automated diagnosis of distributed systems performance problems. In: Technical Report, HP (2006)

Li, H., Zhi, W., Maris, J.: A hidden Markov random field model for genome-wide association studies. Biostatistics **11**(1), 139–150 (2009)

Li, J., Shen, H., Topor, R.W.: Mining optimal class association rule set. In: Proceedings of PAKDD, pp. 364–375 (2001)

Li, P.L., Kivett, R., Zhan, Z., Jeon, S.E., Nagappan, N., Murphy, B., Ko, A.J.: Characterizing the differences between pre- and post- release versions of software. In: Proceedings of ICSE, pp. 716–725 (2011)
Lim, M., Lou, J.G., Zhang, H., Fu, Q., Teoh, A., Lin, Q., Ding, R., Zhang, D.: Identifying recurrent and unknown performance issues. In: Proceedings of ICDM (2014)
Lin, Q., Lou, J.G., Zhang, H., Zhang, D.: iDice: Problem identification for emerging issues. In: Proceedings of ICSE (2016)
Liu, C., Yan, X., Fei, L., Han, J., Midkiff, S.: Sober: statistical model-based bug localization. In: Proceedings of FSE, pp. 286–295 (2005)
Lou, J.G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection. In: Proceedings of USENIX ATC, pp. 24–24 (2010)
Lou, J.G., Lin, Q., Ding, R., Fu, Q., Zhang, D., Xie, T.: Software analytics for incident management of online services: an experience report. In: Proceedings of ASE (2013)
Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B., Zimmermann, T.: Local versus global lessons for defect prediction and effort estimation. IEEE Trans. Softw. Eng. **39**(6), 822–834 (2013)
Nagaraj, K., Killian, C., Neville, J.: Structured comparative analysis of systems logs to diagnose performance problems. In: Proceedings of USENIX NSDI, pp. 271–284 (2012)
Natu, M., Patil, S., Sadaphal, V., Vin, H.: Automated debugging of SLO violations in enterprise systems. In: Proceedings of ICAC, pp. 1–10 (2011)
Patterson, D.A.: A simple way to estimate the cost of downtime. In: Proceedings of USENIX LISA, pp. 185–188 (2002)
Sambasivan, R.R., Zheng, A.X., Rosa, M.D., Krevat, E., Whitman, S., Stroucken, M., Wang, W., Xu, L., Ganger, G.R.: Diagnosing performance changes by comparing request flows. In: Proceedings of USENIX NSDI (2011)
Sharma, B., Chudnovsky, V., Hellerstein, J.L., Rifaat, R., Das, C.R.: Modeling and synthesizing task placement constraints in google compute clusters. In: Proceedings of SoCC (2011)
Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S.C.: A discriminative model approach for accurate duplicate bug report retrieval. In: Proceedings of ICSE, pp. 45–54 (2010)
Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J.: An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of ICSE, pp. 461–470 (2008)
Wong, S.K.M., Ziarko, W., Wong, P.C.N.: Generalized vector spaces model in information retrieval. In: Proceedings of SIGIR, pp. 18–25 (1985)
Yuan, C., Lao, N., Wen, J.R., Li, J., Zhang, Z., Wang, Y.M., Ma, W.Y.: Automated known problem diagnosis with event traces. In: Proceedings of EuroSys, pp. 375–388 (2006)
Zhang, D., Xie, T.: Software analytics in practice: mini tutorial. In: Proceedings of ICSE, pp. 997 (2012)
Zhang, D., Dang, Y., Lou, J.G., Han, S., Zhang, H., Xie, T.: Software analytics as a learning case in practice: approaches and experiences. In: Proceedings of MALETS, pp. 55–58 (2008)
Zhang, D., Han, S., Dang, Y., Lou, J.G., Zhang, H., Xie, T.: Software analytics in practice. IEEE Softw. **30**(5), 30–37 (2013)
Zhang, S., Cohen, I., Goldszmidt, M., Symons, J., Fox, A.: Ensembles of models for automated diagnosis of system performance problems. In: Proceedings of DSN, pp. 644–653 (2005)