Mining Historical Issue Repositories to Heal Large-Scale Online Service Systems

Rui Ding, Qiang Fu, Jian-Guang Lou,
Qingwei Lin, Dongmei Zhang
Microsoft Research
(juding, qifu, jlou, qlin, dongmeiz)
@microsoft.com

Tao Xie
Department of Computer Science
University of Illinois at Urbana-Champaign
taoxie@illinois.edu

***Abstract*—Online service systems have been increasingly popular and important nowadays. Reducing the MTTR (Mean Time to Restore) of a service remains one of the most important steps to assure the user-perceived availability of the service. To reduce the MTTR, a common practice is to restore the service by identifying and applying an appropriate healing action. In this paper, we present an automated mining-based approach for suggesting an appropriate healing action for a given new issue. Our approach suggests an appropriate healing action by adapting healing actions from the retrieved similar historical issues. We have applied our approach to a real-world and large-scale product online service. The studies on 243 real issues of the service show that our approach can effectively suggest appropriate healing actions (with 87% accuracy) to reduce the MTTR of the service. In addition, according to issue characteristics, we further study and categorize issues where automatic healing suggestion faces difficulties.**

***Keywords*-Online service system; healing action; issue repository; incident management**

I. Introduction

Online service systems such as online banking, e-commerce, and email services have been increasingly popular and important nowadays, with an increasing demand on the availability of services provided by these systems. While significant efforts have been made to strive for keeping services up continuously, studies [1] on a sample of hosts have shown that daily and weekly service downs still appear commonly in online services. A serious service down for a non-trivial period often results in huge economic loss or other serious consequences. For example, customers of a service provider may turn to competing providers if the offered services are not available.

In practice, services are often continuously monitored to detect service issues by checking whether service quality violates one of a set of strict pre-defined rules. When a service issue is detected, engineers are called to resolve the issue to pro-actively prevent the issue's potential impact to user-perceived service availability. Thus, reducing the MTTR (Mean Time to Restore) of a service remains one of the most important steps to assure the user-perceived availability of the service [2]. In order to reduce MTTR, a common practice is to restore the service by identifying and applying an appropriate healing action [3] (i.e., a temporary workaround action, such as rebooting a SQL machine) after the occurrence of an issue. Then, after service restoration, identifying and fixing of underlying root causes for the issue are conducted via offline postmortem analysis. In other words, directly applying of an appropriate healing action for the issue wins time for offline diagnosis and fixing of underlying root causes (which typically take relatively longer time to resolve).

However, manually identifying an appropriate healing action for a given new issue is time consuming and error prone. Such manual process is based on investigating service-instrumented data such as transaction logs. According to an internal study from an online service team, about 90% time of MTTR is spent on manual effort for identifying an appropriate healing action. Such substantial manual effort is due to two factors. First, investigating a large amount of service-instrumented data is time consuming. Second, understanding the issue and providing appropriate healing action are heavily depends on domain knowledge. For example, each machine in a real-world product online service (studied in our evaluation in Section IV) produces about 6,000 lines of transaction logs per minute on average. Operators need to inspect these logs from several (usually 4 to 12) machines, and understand the symptom of the issue by reading and reasoning the detailed log information.

To address high cost and error proneness of manually identifying an appropriate healing action, in this paper, we present an automated mining-based approach for suggesting an appropriate healing action for a given new issue. Our approach generates a signature for an issue from its corresponding transaction logs and then retrieves historical issues with similar signatures from a historical issue repository. The historical issue repository records the solved historical issues. Each

issue has a number of basic attributes: affected time, affected location (e.g., specific cluster, network, or datacenter), real customer impact measurement, corresponding transaction logs, etc., along with the appropriate healing action taken by operators to heal the issue. Finally, our approach suggests an appropriate healing action by adapting healing actions of the retrieved historical issues. In particular, our approach measures the similarity between the transaction logs of the given new issue and the transaction logs of a historical issue, by addressing two major challenges due to the high-correlation phenomenon and the weak-discrimination phenomenon. The high-correlation phenomenon refers to the correlation of event's occurrences in transaction logs for causing ineffective historical-issue retrieval. The weak-discrimination phenomenon refers to noisy events that appear relatively independent to the transaction status (being in an issue state or compliant state). Detailed examples for these phenomena are illustrated in Section II.

To tackle challenges due to these phenomena, we develop the technique of concept analysis to address the high-correlation phenomenon and the technique of contrast analysis to address the weak-discrimination phenomenon. Then we define a novel similarity metric to measure similarity between issues and retrieve similar historical issues from the historical issue repository for the given new issue. Finally, we develop a technique of healing-suggestion adaptation to use predefined rules to analyze and adapt the healing actions of the retrieved historical issues to derive a healing suggestion for the given new issue.

In particular, our technique of concept analysis uses Formal Concept Analysis (FCA) to obtain the concept lattice, where highly-correlated events are grouped together as the intents [4] of concepts. Our technique of contrast analysis analyzes the complementary set of events between concepts directly linked through the obtained concept lattice. Such analysis produces the complementary sets of events highly correlated to the transaction status.

Our technique of healing-suggestion adaptation defines the verb + target + location structure to represent a healing suggestion. Both the verb + target are extracted from the retrieved historical issues for the given new issue, whereas the location is extracted from transaction logs of the given issue directly. The verb denotes a specific action from the healing action of the retrieved historical issues, such as "recycle" and "restart". The target denotes a service role from the healing action of the retrieved historical issues, such as "Application Pool", "IIS (Internet Information Service)", and "SQL". The location denotes the affected location of the given new issue, such as "Asia/Network2/Farm332/SQL412-002".

We have deployed our healing system on one online service (serving millions of online customers globally) for more than half a year. During this period, 76 operators in this product team effectively diagnosed and healed this online service with the intensive assistance of our healing system. Our evaluations on this real-world online service demonstrate the effectiveness of our approach in real practice.

To further evaluate the capability, and potential limitations of our approach, we randomly sampled 400 real service issues, and carefully studied our results by simulation. We found that our approach cannot work properly at 157 (39%) issues. We summarize the underlying reasons, which shed lights towards service auto-healing. In summary, this paper makes the following main contributions [1]:

- We formulate the problem of suggesting healing actions for a newly occurred issue as retrieving similar resolved issues in the history, and our techniques of concept analysis and contrast analysis help address challenges and achieve high accuracy on historical-issue retrieval.
- We evaluate our approach on the 243 issues that occurred in the real-world online service in 2012. The results show that our approach can effectively suggest correct healing actions to reduce the MTTR of the service.
- We summarize our experience of applying our approach on the real-world service, investigating issue characteristics and cases where automatic healing suggestion faces difficulties in practice.

The paper is organized as follows. Section II presents examples. Section III presents our approach. Section IV presents evaluation results. Section VI discusses related work. Section VII concludes.

II. Examples

Transaction logs are printed during system execution. Figure 1 shows a log stream collected for two example transaction instances (within the occurring period of an issue): the user-login transaction instance (the highlighted log entries sharing the same transaction ID) and file-editing transaction instance (the un-highlighted log entries sharing the same transaction ID). Such logs record relatively detailed information about run-time behaviors of a system.

[1]This paper significantly extends the previous version of this work (a 4-page ASE 2012 short paper [5]) in three main ways. First, we provide theoretical analysis to validate the effectiveness of our approach. Second, we conduct concrete evaluations of our techniques in a real product environment for over the year of 2012. Third, we further investigate various types of issues that out current approach fails on, shedding light on future directions of improvement.

time	event	transaction ID	message
02/08/2012	a	9d959c…	Request # entering …
02/08/2012	b	9d959c…	created cookie handler with…
02/08/2012	a	7d467b…	Request $ entering …
02/08/2012	c	9d959c…	cookie with name '*' was read …
02/08/2012	x1	7d467b…	SQL server * failover detected…
02/08/2012	x1	7d467b…	SQL server * failover detected…
02/08/2012	x2	7d467b…	$ is not sign…
02/08/2012	x3	7d467b…	Building authentication url
02/08/2012	d	9d959c…	Site=/*/*/…
02/08/2012	x4	7d467b…	attempt to create a sign…
02/08/2012	e	9d959c…	Detected use of * from …
02/08/2012	b	7d467b…	created cookie handler with…
02/08/2012	x5	7d467b…	writing cookie of $
02/08/2012	x6	7d467b…	cookie of $ was not present.
02/08/2012	y1	9d959c…	SqlException: server * was not …
02/08/2012	z	9d959c…	# leaving monitored scope of …
02/08/2012	x7	7d467b…	$ does not require ssl.
02/08/2012	x8	7d467b…	redirecting $ to …
02/08/2012	x2	7d467b…	$ is not sign…
02/08/2012	z	7d467b…	$ leaving monitored scope of …

Figure 1. Log stream for example transaction instances

```
ULS.Logging(
 0x12f3ce22 /* y1*/, Database, "{0}", String.Format(CultureInfo.InvariantCulture,
"SqlException: '{0}' was not found. Source: '{1}' Procedure: '{2}', LineNumber: '{3}' …")
```

Figure 2. Example logging statement in source code

Each log entry (as shown in Figure 1) typically consists of four fields. The log time indicates when the log entry occurs. The event ID is used to identify the corresponding logging statement in the source code. Figure 2 illustrates the corresponding portion of source code for event ID (in short as event throughout the rest of the paper) *y1*, which describes that a SQL exception has been thrown. The transaction ID is used to identify the corresponding transaction instance. The text message describes the detailed runtime information.

In addition, a transaction instance in an online service system has one important attribute: http-status, used by us to determine the transaction instance's fail/success label (see Section III.A.2 for details). The http-status indicates the returned status of a given transaction instance, e.g., "200" denoting "OK" while "500" denoting "Internal Server Error".

The event sequence collected for a transaction instance can reveal part of the code path executed when serving the transaction instance, e.g., revealing which functions are executed. Figure 3 shows sequences of events and their statistics for three transaction types within the occurring period of an issue (including the "file editing" and "user login" types for the two example transaction instances shown in Figure 1 and the "file reading" transaction type).

transaction types	sequence of event	# transactions	
		fail	success
file editing	a, b, c, d, e, y1, z	187	0
user login	a, x1, x1, x2, x3, x4, b, x5, x6, x7, x8, x2, z	36	1
file reading	a, b, c, d, e, z	0	485

Figure 3. Log statistics for three transaction types within the occurring period of an issue

By manually inspecting the information in Figure 3 for issue diagnosis, one could notice that the dominating symptom for the issue is event *y1* (indicating that a SQL exception is thrown), because there are 187 failing transaction instances uniquely associated with this event, in contrast to only 36 failing transaction instances uniquely associated with events *x1 – x8* indicating that an invalid cookie is encountered. Then one could suggest a healing action for this issue as *rebooting a SQL machine* (a typical healing action for a SQL-exception symptom), in contrast to *restarting the Internet Information Services (IIS)* (a typical healing action for an invalid-cookie symptom). However, if one would like to develop mining algorithms to automate this issue-diagnosis process, there exist two phenomena on transaction logs for posing challenges.

High-correlation **phenomenon.** We observe that some events always appear together, being highly correlated. The reason for such observation is that the developers want to track execution states with finer granularity at some critical statements, such as the credential-verification session. Such tracked states capture sufficient logging information for diagnosis when causes of issues are related to the execution of these statements. For example, when event *b* appears, *c*, *d*, and *e* always follow (see Figure 3). As another example, events *x1–x8* always appear together to indicate invalid cookies. If we do not group them together when comparing event sequences for the given issue and historical issues, events *x1–x8* would contribute eight times than event *y1* to characterize the given new issue, likely causing this given issue to be wrongly matched with a historical issue with the dominating symptom as events *x1–x8*. Such wrong matching would cause a wrong healing action to be suggested.

Weak-discrimination **phenomenon.** Some events are noisy, being weakly-discriminating events, whose appearance is independent of the transaction status (i.e., issue state or compliant state). Examples of such events are *a* and *z* (in Figures 1 and 3) for indicating entering and leaving actions for each transaction instance, respectively. These events appear in almost every transaction instance. Log messages corresponding to these events contribute little to distinguish different types of issues. Thus, due to such weakly-discriminating events, retrieved historical issues for the given new issue may not be desirable. For example, assume that another different issue from the historical issue repository is dominated by events *a*, *b*, *d*, *y2*, and *z* where "y2" is related to "antivirus timeout". If we do not address such phenomenon, we would wrongly retrieve this historical issue for the given issue related to SQL exception (since the only difference of events for these two issues is "y1" vs. "y2".

III. Approach

Our approach consists of three steps. First, we use concept analysis and contrast analysis to generate the signature for an issue. Second, we retrieve historical issues similar to the given new issue from an issue repository based on their generated signatures. Third, we produce healing suggestions by adapting the healing actions of the retrieved historical issues.

A. Signature Generation

Our approach includes the techniques of concept analysis and contrast analysis to address high-correlation and weak-discrimination phenomena. Concept analysis applies Formal Concept Analysis (FCA) to group highly-correlated events together as the intent of a concept. Contrast analysis calculates Mutual Information to measure the correlation between each concept and its corresponding transaction status, and then evaluates the complementary set of intents between parent and child concepts in concept lattice by measuring their Delta Mutual Information (DMI). We generate the signature for the issue as the complementary sets that satisfy the predefined criterion.

1) Concept Analysis: In our problem, each transaction instance corresponds to an event sequence. However, we ignore the information of temporal ordering and event-recurrence count, and use an event set to represent to each transaction instance. Although the information of temporal order and event-recurrence could indicates particular failure characteristic (e.g., race-condition, and iterating in a loop, respectively), service issues relate to such information are very rarely appeared in practice. So our simplification improves efficiency, while preserve enough effectiveness. Then, we group together highly-correlated events by applying FCA. The intuition is that highly-correlated events together indicate one kind of symptom. FCA is a principled way to automatically group such events together [13].

Figure 4 illustrates two concept nodes from the concept lattice, which is constructed from the logs in Figure 3. The gray node in the middle of Figure 4 is "file-editing" + "file-reading", which is the parent to the gray node in the bottom. More precisely, each concept c contains a set of events, called the intent, denoted by Int(c). The intent of a parent node always belongs to the intent of its child (note that according to FCA theory, the parent-child relationship is constructed by such partial-order relation). In addition, the extent of each concept is a set of transaction instances, denoted by Ext(c). According to FCA theory, the event set belongs to each transaction instance in Ext(c) shares the same Int(c).

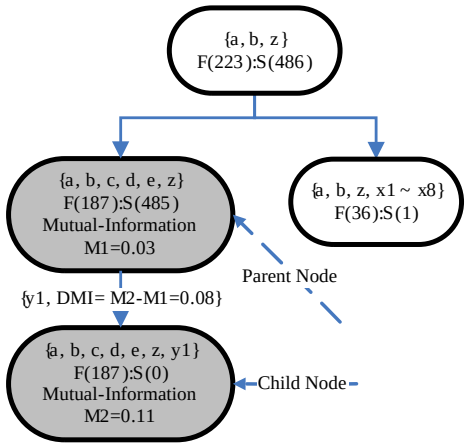

Figure 4. Relationship between two linked concepts in the lattice

Table I
Joint Distribution of X_c, Y

	$Y=1$	$Y=0$
$X_c=1$	x	y
$X_c=0$	$n-x$	$m-y$

2) Contrast Analysis: By leveraging the fail/success information of each transaction instance and the relationship between parent and child concepts, contrast analysis finds the subset of the events that are highly correlated to failed transaction instances. We next give a definition about the fail/success label for a transaction instance, and then present our considerations about positive correlation and delta mutual information.

Fail/success label. We define the label for each transaction instance (reflecting the transaction status) as

$$label_i = \begin{cases} failure, & HttpStatus_i \geq 500 \\ success, & \text{otherwise} \end{cases}$$

where i denotes the index of a specific transaction instance. Note that, although we use such specific definition in our problem, it can be flexibly and easily modified according to the different requirements of different scenarios.

Positive correlation. We calculate mutual information to measure the correlation between a concept and failures.

Let x and y be the number of failed and succeeded transaction instances for a given concept c, respectively; let n and m be the total number of failed and succeeded transaction instances within the occurring period of a given new issue, respectively. Then we define a random variable Y, which indicates the outcome (1 refers to fail, and 0 refers to success) of a randomly selected transaction instance; and another random variable X_c, which indicates the outcome of a randomly selected transaction instance belongs to Ext(c).

Then the outcomes x, y, n, m can approximately represent the joint distribution of X_c and Y as illustrated

in Table I. In our approach, we adapt the formula of mutual information as below (by dropping the $< 10 >$, $< 01 >$ items):

$$M(X_c, Y) = P(X_c = 1, Y = 1) \log \frac{P(X_c = 1, Y = 1)}{P(X_c = 1)P(Y = 1)}$$

$$+P(X_c = 0, Y = 0) \log \frac{P(X_c = 0, Y = 0)}{P(X_c = 0)P(Y = 0)}$$

Thus we use only the positive correlation part of mutual information. In general, the negative correlation ($< 10 >$, $< 01 >$ items) happens trivially often in network traces and are not meaningful [6].

Delta information. To achieve accurate retrieval, we need to exclude noisy events and keep only "clean" events. For example, in Figure 4, the mutual information of the child node is high, but the events {a, b, c, d, e, z} that it contains are irrelevant to the failures, and should be eliminated. To address this problem, we analyze *Delta Mutual Information (DMI)* between child and parent concept nodes in the concept lattice, to measure how the delta events contribute to correlation.

Let $\triangle E_s = Int(chi) \setminus Int(par)$ be the extra events that the child node has, e.g., $\triangle E_s$ being {y1} in Figure 4. Then we define

$$DMI(\triangle E_s) = M(X_{child}, Y) - M(X_{parent}, Y)$$

Intuitively, $DMI(\triangle E_s)$ represents the contribution of the extra events $\triangle E_s$ for failure correlation. By walking through each edge in the concept lattice graph, we select $\triangle E_s$ as a term if it satisfies criteriaX:

$$criteriaX = \begin{cases} M(X_{Int(par)}, Y) > 0 \\ DMI(\triangle E_s) > 0 \end{cases}$$

Intuitively, the first inequality indicates there exists a positive correlation between a concept par and failure, and the second inequality indicates there exists "more" failure-correlation due to the extra events that the child node contains. In addition, Such definition of term has a number of important properties. The theorem below brings a bridge between our criteria and human's intuition when diagnosing the service issues.

Theorem $DMI(\triangle E_s) \approx \frac{\partial M}{\partial x}\triangle x + \frac{\partial M}{\partial y}\triangle y$ where $\frac{\partial M}{\partial x} > 0$ and $\frac{\partial M}{\partial y} < 0$.

Intuitively, let $\triangle E_s$ be a set of events which satisfy $criteriaX$, then the higher value of the $DMI(\triangle E_s)$ means that events in "$\triangle E_s$ appear more probably (higher value of $\frac{\partial M}{\partial x}$) in failed requests, and less probably (lower value of $\frac{\partial M}{\partial y}$) in succeeded requests as well". We do not give the detailed proof in this paper due to space limit. Readers can refer to our project website [7] for the theorem and detailed proof. This property also inspires us for developing further similarity measurement.

In the example in Figure 4, {y1} is a term, since mutual information of the parent node is 0.03, and the corresponding DMI is 0.08. Next, we use the $DMI(\triangle E_s)$ as the weight for term $\triangle E_s$. So a signature S_{issue} is the collection of all the terms, which can be represented as follows:

$$S_{issue} = \{< \triangle E_s, DMI(\triangle E_s) > | criteriaX = true\}$$

B. Similar-Issue Retrieval

In similar-issue retrieval, we first need to have a representation for issues and then define a similarity to measure issue similarity so that the most similar issue for the given new issue could be retrieved. We implement the term-weighting and document-scoring function of Generalized Variable kernel Similarity Metric (GVSM [8]) to measure the defined similarity metric.

1) Issue Representing: We treat each issue in the historical issue repository as one document, each signature as a set of terms, and DMI as the weight of each term. Let $D = \{d_1, d_2, ..., d_m\}$ 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 $\sum_{p \in A(i)} w_{ip}\vec{t_p}$, where a term is represented by an abstract vector $\vec{t_p}$, 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 $\vec{t_p}$. We use DMI as the weight for each term. Such weight is much different from the *TF-IDF* weight [9]. Our evaluations (Section IV) compare the results of such difference.

2) Similarity Metric: We calculate the cosine score of two document vectors, each representing one issue. In particular, given a documents d_i that $d_i = \sum_{p \in A(i)} w_{ip}\vec{t_p}$. We next define the similarity metric

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

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: $\vec{t_p} \cdot \vec{t_q}$ = # of overlapped events in *p*-th term and *q*-th term.

We abandon the orthogonal assumption in the conventional vector space model, since the orthogonal assumption is too restrict (this assumption is also referred to as exact-match: the inner product is equal to 1 if and only if the two sets are exactly the same; otherwise, 0). We aim to give a similarity score between 0–1 rather than yes/no. We prove that our similarity metric satisfies the requirements of GVSM, so our problem can be modeled as a text retrieving problem, and we could leverage the corresponding benefits due to the properties of GVSM. We do not give the detailed proofs here due to space limit. The detailed deductions can be viewed at our project website [7].

Table II
RULE-BASED MAPPING

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	WAC (Web Application)	ev3
patch	Service (SQL/WFE/WAC)	ev1,ev2,ev3
restart	Scanner (Anti-virus Component)	ev2,ev3
restart	Search (Search Component)	ev3
restart	AD (Active Directory)	ev4

Table III
CATEGORIES OF STUDIED HEALING ACTIONS

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

C. Healing-Suggestion Adaptation

We use a triple structure $< verb, target, location >$ to represent a healing action, and manually extract the verb and target from the description of the historical issue retrieved for the given new issue, and extract the location from log messages of the given new issue. The extracted healing actions are reasonably proper, since all these issues are well-resolved, and the corresponding healing actions have been verified according to the incident management process from the product teams.

Based on empirical investigations of healing actions for online service systems, we find that most healing actions can be formatted as $HealingAction = verb + target + location$. A verb is an action, such as "reboot" and "re-image" (re-image: to completely replace the operating system with a pre-configured image). The major types of verbs in our problem setting are illustrated in Table II. A *target* represents a component or a service role, such as an Internet Information Services (IIS) or a database, as illustrated in Table II. A *location* is an exact affected machine name with its physical location. When we retrieve a similar historical issue for the given new issue, we obtain "verb" and "target" from the historical issue, e.g., from the description text "*We found few SQL servers with high memory usage and few servers were not able to connect through . Availability is back up after* ***rebooting*** *the* ***SQL machine*** *SQL32-003*", we extract the verb as "reboot" and the *target* as "SQL". The combination of a verb and a target is not arbitrary; Table II shows all the possible combinations according to our study.

We extract the *location* (the specific machine names) with a rule-based technique. The specific machine name is typically mentioned in the log messages associated with a fixed set of events. For example, the log message "*Cannot connect to SQL server* * ..." is always associated with event ev1, so we find event ev1 from logs and then extract * from the log message as *location* by using regular expression. The complete mapping is illustrated in Table II.

Note that manually identifying an appropriate healing action for the given new issue is typically non-trivial since not only identifying an appropriate healing action needs high-confidence evidence but also there are totally 9 types of healing actions (as listed in Table II), which could be instantiated to form a non-trivial number of possible healing actions in the search space. Hence inspecting our suggested healing action(s) (e.g., the healing action identified from the top-k most similar historical issues) could apparently reduce the time-cost on the two aspects.

IV. EXPERIMENTAL EVALUATION

In our evaluations, we intend to answer three research questions:

- *RQ1.* How effectively can our approach suggest appropriate healing actions for the given new issues?
- *RQ2.* How well does our technique for addressing the high-correlation phenomenon contribute to the overall effectiveness of our approach?
- *RQ3.* How well does our technique for addressing the weak-discrimination phenomenon contribute to the overall effectiveness of our approach?

A. Experiment Setup

We evaluate our techniques in a released production service, named ServiceX. ServiceX is a customer-facing, geographically distributed, 24 x 7, 3-tier online service, with five datacenters around the world. We next describe the collection of trace data and definition of evaluation metrics for our evaluations.

We randomly sampled 400 issues from the resolved issues which are detected by the internal monitoring system in the year of 2012. Among these issues, 243 issues are with clear resolutions and transaction logs, and these issues are valid for our evaluations.

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 III. In the categorization, we do not consider or list the location information in the healing actions since it is different across different issues.

1) System Topology: The ServiceX system includes five datacenters. Each datacenter can be represented by a hierarchical topology, with many "*farms*" as the leaf node. Each *farm* is a unit of full functionality for

serving a set of customers, which contains servers of multi-roles (e.g., WFEs, SQLs). The transaction logs of each individual server are temporarily stored in the corresponding local machine for a certain period of time (the detailed information cannot be exposed due to Microsoft confidential). According to a topology-manager, which maintains the mapping from logical server role to the specific physical machine, we can query the corresponding transaction logs by given a specific server name. The service issues are triggered by monitoring system as well as real customers. When an issue is triggered, it is sent to a global historical issue repository named as RepX. Each issue in RepX is associated with an issue ID, affected farm, and time period, etc. Such associated information is used to identify transaction logs for the issue.

2) Metrics: To comprehensively evaluate our approach, we design two-scenario strategies which mimic the two major real usages of our technique. In both scenarios, we in-turn treat each of the 243 issues (in the order of their occurring time) as a "new issue". But the two scenarios differ in what issues we choose as the "historical issues".

In Scenario I, 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 approach's effectiveness in suggesting a correct healing action for the "new issue".

In Scenario II, we adopt the "leave-one-out" strategy (a common strategy used in statistical analysis) by treating all the *remaining* issues (other than the "new issue") as the "historical issues". Scenario II is used for building a knowledgebase which manages all historical issues.

Note that in our results, the *location* info for a suggested healing action is always correct for each issue, because only an unhealthy service would produce ev1 – ev4 (see Table II). Therefore, the retrieval accuracy is critical in the overall effectiveness of our approach: if the healing action for the retrieved similar historical issue for a "new issue" is in the same category as the correct healing action for the "new issue", then the healing suggestion is correct (since the *location* information of the healing action is assured to be correct as described earlier).

3) Experiment Design: To answer the second and third research questions, we apply two approaches (variants of our approach) in short as *App1* and *App2* besides applying our approach in short as *Ours*. In *App1*, we do not address the high-correlation phenomenon: we calculate Mutual Information of each individual event as its weight (using contrast info), then represent the events as a vector, and finally calculate the cosine score as the similarity metric value. In *App2*, we do not address the weak-discrimination phenomenon: we first apply FCA and use delta events between parent and child concepts to define terms (using grouping information), use TF-IDF as the weight of each term, and finally calculate the cosine core as the similarity metric value.

Detailed Design of Scenario I: We design an experiment that estimates the accuracy of top 1 similar issue being retrieved by our approach to mimic one main scenario when our technique is used in real practice. We first sort all the issues by the occurring time from the earliest to the latest. Then for each approach, we initialize the score as zero. Then for each "new issue" q_i, we check the top 1 similar "historical issue" (here the historical issues refer to the issues that occurred earlier than q_i) retrieved by our approach: if the retrieved "historical issue" belongs to the same category of healing actions as the "new issue", we increase the score by one. At the end of all iterations, for each q_i, we can attain an average score, which reflects the average accuracy of healing suggestion at the time point of q_i. We draw such curve with X-axis being the index of the sorted issues and Y-axis being the average accuracy at the time point of the corresponding q_i.

In Scenario I, we also evaluate the cost performance of our approach. The runtime cost of our approach consists of two parts: signature generation and retrieval. In the part of signature generation, we apply our signature-generation algorithm to only a new issue. After the signature is generated, we store it into text indexed by issue IDs. In practice, loading signatures of historical issues is very fast, so we can ignore the loading time. In the part of retrieval (i.e., similarity calculation), as the number of historical issues grows, the time complexity of this part grows linearly.

Detailed Design of Scenarios II: In our evaluations, for each combination of each approach (Ours, App1, and App2), each "new issue" q_i selected from healing-action category cat_k, and a given similarity threshold s, we measure the precision for our approach's effectiveness in suggesting correct healing actions:

$$pre(q_i, s) = \frac{\#\text{retrivals in } cat_k(similarity > s)}{\#\text{retrievals}(similarity > s)}$$

Then we measure average precision for cat_k:

$$pre(cat_k, s) = average(pre(q_i)), \forall q_i \in cat_k$$

We measure the average precision for all categories:

$$pre = average(pre(cat_k, s)), \forall cat_k$$

In our evaluations, we set s from 0.6 to 0.995, with 0.05 as each increment step. Then we get the

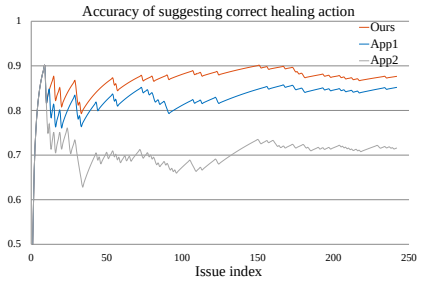

Figure 5. Accuracy of suggesting correct healing actions

highest precision for our approach, App1, and App2, respectively.

In our target problem, recall is not a useful metric, since the decision of the final healing action is typically not based on recall. According to the feedback from engineers (short for product engineers, whose duties are service diagnosis and recovering), they mainly care about the retrieved candidates themselves, without caring about the size of the categories that the specific issues belong to, so we do not use recall as an important metric.

B. Experiment Results

We next illustrate our experimental results of Scenarios I and II, respectively.

1) Results of Scenario I: Experiments in Scenario I address the accuracy of suggested healing actions in real scenarios. Figure 5 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.

RQ1: Approach Effectiveness: The overall accuracy of our approach, App1, and App2 is 87%, 82%, and 72%, respectively. Achieving the best effectiveness, our approach correctly suggests healing actions for 213 issues. Figure 5 shows the trend of the average accuracy: the curve of our approach is always on top of curves of App1 and App2.

Note that there are at least 9 issues for which wrong healing actions would be suggested, since these issues are the first issue of each of the total 9 categories, and no previously encountered issue of the same category is available for them to leverage. Such cases are represented by some points (e.g., the value of X-axis is 11, 17, 32) located at sharp drops in Figure 5.

Beside these 9 issues, our approach wrongly suggests healing actions for 21 issues. We provide further investigation on these issues in Section IV.C.

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.

Table IV
APP1 SUGGESTS CORRECT HEALING ACTIONS, WHEREAS OUR APPROACH DOES NOT

Issue#	Top1_Similarity (Ours)	Top1_Similarity (App1)
153	0.02	0.27
178	0.24	0.24

Table V
OUR APPROACH SUGGESTS CORRECT HEALING ACTIONS, WHEREAS APP1 DOES NOT

Issue#	Top1_Similarity (Ours)	Top1_Similarity (App1)
14	0.91	0.49
45	0.68	0.97
79	0.82	0.87
83	0.71	0.28
86	1.00	0.61
90	1.00	0.96
91	0.60	0.75
124	1.00	0.74

RQ2: Concept-Analysis Effectiveness: The blue-colored (middle) curve in Figure 5 is for App1. Our approach's curve is on top of it at each value of X-axis. App1 correctly suggests healing actions for 207 issues, whereas our approach correctly suggests healing actions for 6 more issues in total.

Table V and IV list all the issues with different suggestions between our approach and App1 (including the issues that our approach gives correct suggestions but App1 does not, and those vice versa). There are 2 issues that App1 gives correct suggestions (the first 2 rows), and 8 other issues that our approach gives correct suggestions (the last 8 rows).

To understand the reasons for such different suggestions, we conduct further investigation. Table V and IV further lists the top1 similarity score for each issue, computed by our approach and App1, respectively. We can observe that the 2 issues that App1 performs better are trivial; the similarity score there is low: 0.27 is the highest score. Such low score indicates that most parts of the signatures between the current issue and the top1 similar issue are not that similar (recall that in the experiment design of Scenario II, we set the similarity threshold to 0.6 as the lower bound). Further investigation of the detailed log events and messages confirms with such observation: in fact, the two issues (#153 and #178) are "outliers" compared with other issues, although correct healing actions are suggested for them in the end.

On the other hand, all the 8 issues that our approach performs better have at least 0.6 of top 1 similarity score, and most scores are even close to 1.0. Further

investigation of these issues confirms that the current issue and the most similar issue are indeed related, App1 does not suggest correct healing actions because of its main weakness: the bias of several terms of large size (i.e., a signature consisting of a large number of events).

One typical example is the issues in category ID7, each describing a specific service trouble named as an "ADO.NET" issue. These results show that our approach's overall effectiveness benefits from addressing the high-correlation phenomenon. Dominating terms (i.e., those with significantly larger weight than the other terms of the same issue) of this category are {x} and {y1 y2 y3 y4 y5}. When the execution of a transaction instance goes through event {x} and {y1 – y5}, such issue of a swift timeout can be reproduced. However, the terms of many issues in the category of "SQL resource" (ID4) are {z} and {y1 y2 y3 y4 y5}, with only one event being different: "z" instead of "x". Such difference is small in quantity but is impactful. App1 can hardly distinguish an issue from the category of "ADO.NET" from an issue from the category of "SQL resource", and would report a high similarity score for these two issues, leading to wrong healing suggestions. However, in our approach, set {y1 – y5} contribute the same weight as {x}, so the final similarity score would not be biased by a specific term of large size. Note that the healing actions for the two categories "ADO.NET" and "SQL resource" are different, being patching a machine and rebooting an SQL machine, respectively (see Table III).

RQ3: Contrast-Analysis Effectiveness: The green colored (bottom) curve in Figure 5 is for App2. App2 correctly suggested healing actions for 174 issues, with its accuracy as about 72%. Our approach improves the accuracy of App2 by about 21%, which is substantial. In summary, the evaluation result shows that, considering contrast information (i.e., fail/success of each request) substantially contributes to the overall accuracy of our approach.

Runtime Performance of Our Approach: We generate the signatures of the total 243 issues with the runtime cost of 45,848ms; thus, on average, we generate the signature for each issue with the runtime cost of about 189ms.

Figure 6 shows the run time cost of retrieving the top1 historical issue for each issue. The x-axis shows the issue index, and the y-axis shows the time cost (with unit as ms). We can see that the speed of processing the first 150 issues is really fast (less than 50ms). As the issue index grows, the runtime cost of the retrieval grows, yet being still small (less than 250ms). According to the experiences from engineers, in practice, less than 1 minute is already an acceptable bound for healing services, since it is much less than the common MTTR (the actual value of MTTR of ServiceX is not exposed due to Microsoft confidential).

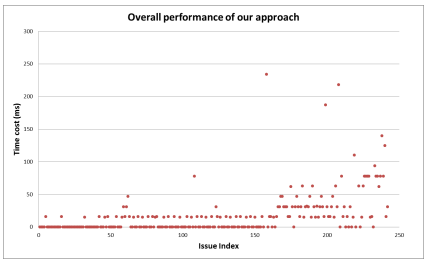

Figure 6. Overall performance of our approach

Table VI
OVERALL PRECISION

	similarity threshold	highest precision
Ours	0.85	0.87
App1	0.81	0.81
App2	0.94	0.58

The signature-generation part of App1 is 100 times faster than the signature-generation part of our approach, since App1 calculates the mutual information of only each individual event; however, the retrieval part of our approach is 5 times faster than the retrieval part of App1. One major reason is that our contrast analysis eliminates most of irrelevant events; however, in App1, the comparison of the new issue and a historical issue can involve hundreds of unique events including many irrelevant ones.

2) Results of Scenarios II: We use the similarity threshold s as a parameter to get the $pre\ s$ curve of each approach. This part of evaluation complements to the evaluation of Scenario I. To make the comparison fair, we consider only the highest precision of each approach.

Table VI shows that the highest precision of our approach is 0.87, with the corresponding similarity threshold as 0.85. We can see that our approach also performs the best compared to App1 and App2 in terms of precision.

Note that high precision is very desirable for another real scenario in practice. In particular, first, engineers would like to set a similarity threshold. Then, after a new issue occurs, they would like to see all the possible historical issues (compared to the new issues) with similarity exceeding this threshold. Finally, the engineers inspect these retrieved historical issues and make the final decision on which healing action to apply. In this scenario, higher precision allows the engineers to gain higher confidence in making decisions based on the retrieved historical issues.

C. Experiences in Real Product

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 suggest correct healing actions. We

Table VII
ISSUES WHERE OUR APPROACH IS NOT APPLICABLE OR FAILS

Inapplicable issues			Issues our approach fails	
Issue type	Count		Issue type	Count
No logs available	78 (19.5%)		one-shot issue	12
Service upgrade	31 (7.8%)		logs not enough	4
Auto recovered	28 (7.0%)		latency issue	3
False alarms	5 (1.3%)		insufficient events	2
Other	15 (3.8%)		Total	21

further improve our algorithm design and implementation based on some of these findings. In this section, we discuss these experiences focusing the types of new issues where our approach is not applicable and the types of new issues where our approach fails to suggest correct healing actions.

1) Issues Where Our Approach is Not Applicable: We systematically and manually investigate the sampled 400 issues in RepX (see Section IV.A). Besides the 243 issues used in our evaluations, the remaining 157 (39%) issues do not satisfy our input requirements. The statistics on types of these issues are listed in the left-hand side of Table VII. We next illustrate the four main types of issues.

The type of "No logs available" describes the issues where we cannot collect corresponding transaction logs anywhere. The underlying reasons vary: some were due to network issues, so the user requests did not reach the application service; some were due to changes of the topology in the system, so that the alerted service no longer existed when engineers started the investigation.

The type of "Service upgrade" describes issues that are "noisy" issues alerted during system upgrade, i.e., the monitoring system was not shut down in time when the service upgrade began. Engineers would just leave a note of "*this is trivial alarm that don't need to investigate*" and close it, so we cannot obtain corresponding healing actions.

The type of "Auto recovered" describes issues that were automatically recovered before engineers started the investigation.

The type of "False alarms" describes issues that were filed due to cases not related to real service issues, e.g., some internal testing bugs were wrongly filed as service issues by mistake.

2) Issue Where Our Approach Fails: We study the 21 issues that our approach does not suggest correct healing actions, and categorize them in the right-hand side of Table VII.

The type of "one-shot issue" (12 issues) describes issues with unique signatures, which are not similar to the signatures of any other issues (each of the 12 issues is not similar with each other either). According to the feedback on these issues from engineers, the logs of these issues provide useful information for diagnosis, and the signatures that our approach generates are still valuable to the engineers in diagnosis. Our approach fails on these issues because there exist no similar historical issues for these issues.

The type of "logs not enough" (4 issues) describes issues with log information insufficient for diagnosis. Engineers did not identify the root causes by inspecting these logs, and our approach does not generate helpful signatures either. According to the discussion with engineers, the logging was not sufficient when the system executed the code paths that lead to these 4 issues. On the other hand, 4 is a small number, implying that the current logging practice is generally good enough.

The type of "latency issue" (3 issues) describes issues where the user requests are processed with suspiciously long time, but are still processed successfully. Users would feel unhappy about such slow response; however, our approach cannot handle such long-latency issues well, because the unhappy (long-latency) code paths are not different to the happy (fast) code paths, and no events for these two cases are discriminative.

The type of "insufficient events" (2 issues) describes issues that are associated with insufficient events. Using only events or event sets associated with these issues cannot well discriminate the issues with other issues. More information from log messages should be leveraged to generate more proper signatures. For example, from the system's 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 to indicate different system failures, which could lead to different healing actions. So using only this event cannot well discriminate different issues. However, since the exception information is recorded in the message column, our approach's effectiveness can be improved by simply combining the event with its partial message to construct a more useful new event.

D. Threats to Validity

The threats to external validity primarily include the degree to which the studied online service, its issues, its usages, etc. are representative of true practice. The studied online service is a real-world product online service that serves millions of customers globally. The investigated issues and service usages come from real-world cases. These threats could be reduced by more experiments on wider types of subjects in future work. The threats to internal validity are instrumentation effects that can bias our results. Faults in our healing system might cause such effects. To reduce these threats, we manually inspected trace data and our system outputs for a number of issues.

V. Real Case Studies

In this section, we select two typical real issues occurred in one online service system, to demonstrate the effectiveness and potential capability of our approach.

Antivirus Configuration Corruption.

ServiceX experienced continuous performance problem in one datacenter in January of 2012. During the occurrence of this issue, customers experienced both slow response and failed to upload files in an unpredictable fashion. Operators who first diagnosed this issue found that one Web Front End (WFE), named WFEx, "produced" most *http-status* = 500 failures. Since each transaction instance randomly selected one WFE to be served due to the load-balancing strategy, only the transaction instances that go through WFEx would have a high probability to fail. The operators asserted that WFEx went into a bad state, so they rebooted it after investigation. However, such rebooting action did not turn back the service availability, and the same issue existed continuously. After involving senior experts, they finally found the root cause of the issue to be that the "*configuration file for antivirus software became corrupted after a random restart*". In the end, the only resolution was to re-image.

Resolving this issue is challenging, involving 12 experts in different relevant teams and with mail discussion in about two weeks to find the root cause by investigating various logs of different components/features. This issue is also a tricky one since people usually reboot a WFE after general diagnosis. Our approach finds the symptoms (denote as ACC: Antivirus Configuration Corruption) "antivirus timeout" (which led to "Internal Server Error", reflecting that users failed to upload files) and "SQL failing over detected" (which led to long latency, reflecting that users felt slow response) on only WFEx. We 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 with 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 operator, who was not familiar with this issue, immediately moved to check the antivirus configurations instead of rebooting the WFE (a common healing action). Our approach helped reduce much investigation time of the operator by providing informative diagnostic clues. After repeated occurrences of such issue with the ACC symptom, the issue was finally marked as a "need fix" issue of antivirus software, and is to be fixed in the future upcoming service upgrade.

False Alarms of Monitoring System.

We provide another interesting story that occurred after we conducted the evaluation. This story demonstrates that not only the healing actions, but also the information about descriptions and diagnostic steps of similar historical issues could be leveraged to help investigate current issues.

Starting from early July of 2012 till the end of the month, the monitoring system of ServiceX sent out 120 specific types of alarms (service issues). These issues were not easy to diagnose. According to the email discussion from the involved operators, and also the information recorded in the issue repository, the involved operators did observe some new suspicious transaction logs in the effected machine; however, these machines seemed running healthily without any abnormal behavior. To be conservative, operators had to inspect the potential dependent machines one by one. After hours of investigation, operators had no findings, and had to temporally tag "no clue, postponed" to the issue in the issue repository.

This challenging issue had been resolved about three weeks later with more than ten experts involved. The root cause was that the monitoring system reported a lot of false alarms due to the incompatible versions of components (of the monitoring system) after the previous upgrade. All the 120 false issues were then marked as "duplicated" in the repository. Similar issues never occurred again after the subsequent service upgrade.

We captured and replayed the whole story by simulation. According to our simulation study, we found that if the first three issues (which recorded detailed descriptions and diagnostic steps, and occurred in the first two days) were labeled as a similar group, all the remaining 117 issues could be retrieved correctly (i.e., for each unlabeled issue, the most similar one is one of the three, and with the similarity-metric value $>$ 0.90). Although there were no healing actions associated with these issues, the rich information of the previous investigations (i.e., those for the first three issues) can substantially reduce redundant efforts for diagnosing the large number of recurrent issues.

VI. Related Work

We discuss related work in the areas of system diagnosis, fault detection, and mining software repositories.

System diagnosis. Cohen et al. [10] propose that retrieving a previously solved and annotated issue similar to the given new issue may help identify the root cause or fixing action for the given issue when the retrieval is accurate. In contrast, rather than aiming to fix the issue, our work aims to provide healing suggestion to reduce MTTR by leveraging historical issues. Yuan et al. [11] use classification techniques to classify issues into different categories. In contrast, our work does not require specific labeling but retrieves similar historical issues for adapting their healing actions as suggested healing

actions for the given issues. Previous work [12], [13] on automated diagnosis of distributed systems uses two types of trace data for analyzing system performance: system metrics/events and transaction logs. Our work requires only transaction logs for signature generation and healing suggestion.

Fault localization. Our technique used to generate signature from transaction logs shares a similar high-level concept with the fault-localization technique proposed by Liu et al. [14]. Sun et al. [15] evaluate patterns mined from both correct and incorrect runs to detect duplicate bug reports. Our work uses contrast information for achieving high accuracy of signature generation. Cellier [16] applied FCA for fault localization by using concepts to find interesting clusters. In contrast to these previous techniques on fault localization based on coverage information, our work is motivated by addressing challenges posed by characteristics of transaction logs.

Mining bug repositories. When a new software fault is reported, operators usually use a web search engine to search for the text of error messages or console messages for diagnosis. The essence of such scenario remains when the web search engine is replaced by a search engine for a bug repository. Ashok et al. [17] implement a search tool to speed up bug fixing by leveraging natural language text, dumped traces, and bug output. Some other work [15], [18] 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 in a typical historical issue repository is incomplete or imprecise.

VII. Conclusion

To effectively reduce the MTTR of a service, we have proposed an automated mining-based approach for suggesting an appropriate healing action for a given issue. Our approach suggests an appropriate healing action by adapting healing actions for the retrieved similar historical issues. Our studies on a real-world product online service show that our approach can effectively provide appropriate healing actions to reduce the MTTR of the service.

References

[1] D. D. E. Long, A. Muir, and R. A. Golding, "A longitudinal survey of internet host reliability," in *Proc. SRDS*, 1995, pp. 2–9.

[2] W. Xie, H. Sun, Y. Cao, and K. Trivedi, "Modeling of online service availability perceived by web users," in *Proc. GLOBECOM*, 2001.

[3] D. Cannon and D. Wheeldon, "The stationery office," in *ITIL Service Operation*, 2009.

[4] G. Bernhard, S. Gerd, and W. Rudolf, "Formal concept analysis: Foundations and applications," in *LNCS*, 2005.

[5] R. Ding, J. J. Shen, Q. Fu, J. G. Lou, Q. Lin, D. Zhang, and T. Xie, "Healing online service systems via mining historical issue repositories," in *Proc. ASE*, 2012.

[6] S. Kandula, R. Chandra, and D. Katabi, "What's going on?: Learning communication rules in edge networks," in *Proc. SIGCOMM*, 2008, pp. 87–98.

[7] (2014, Feb.) Appendant materials. [Online]. Available: http://research.microsoft.com/en-US/people/juding/kb.aspx

[8] G. Tsatsaronis and V. Panagiotopoulou, "A generalized vector space model for text retrieval based on semantic relatedness," in *Proc. EACL*, 2009, pp. 70–78.

[9] H. Wu, R. Luk, K. Wong, and K. Kwok, "Interpreting tf-idf term weights as making relevance decisions," in *TOIS*, 2008, pp. 1–37.

[10] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *Proc. SOSP*, 2005, pp. 105–118.

[11] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma, "Automated known problem diagnosis with event traces," in *EuroSys*, 2006, pp. 375–388.

[12] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *Proc. OSDI*, 2004, pp. 231–244.

[13] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *Proc. EuroSys*, 2010, pp. 111–124.

[14] C. Liu, X. Yan, L. Fei, J. Han, and S. Midkiff, "Sober: statistical model-based bug localization," in *Proc. ESEC/FSE*, 2005, pp. 286–295.

[15] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proc. ICSE*, 2010, pp. 45–54.

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

[17] B. Ashok, J. M. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala, "Debugadvisor: A recommender system for debugging," in *Proc. ESEC/FSE*, 2009, pp. 373–382.

[18] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proc. ICSE*, 2008, pp. 461–470.