

A Methodology for Root-cause Analysis in Component Based Systems

Kui Wang^{*§}, Carol Fung[†], Chao Ding[§], Polo Pei[‡], Shaohan Huang[§], Zhongzhi Luan^{*§}, Depei Qian^{*§}

^{*}Beijing municipal key laboratory of network technology, Beihang University, Beijing, China

[†]Computer Science Department, Virginia Commonwealth University, Richmond, Virginia, USA

[§]Sino-German Joint Software Institute, Beihang University, Beijing, China

[‡]Tencent Corporation., Shenzhen, China

kwang@buaa.edu.cn, cfung@vcu.edu, super.d.scse@gmail.com,

polopei@tencent.com, buaahsh@foxmail.com,

zhongzhi.luan@jsi.buaa.edu.cn, depeiq@buaa.edu.cn

Abstract—In component based enterprise systems, anomaly detectors are commonly deployed on application-level components, but not on lower-level functional components. When anomaly alarms are triggered, system managers are expected to handle them in a timely manner to avoid cascading failures. Excessive large volume of anomaly alarms makes them impractical to handle manually. Most existing root cause analysis methods are based on the assumption that all components are monitored and analysis are performed based on the time correlation of the generated alarms. However, full monitoring coverage may not be practical due to cost and complexity. In this paper, we present RCSF, a root cause analysis method that targets at systems where only application-level components are monitored by anomaly detectors. The method analyzes the components performance log on functional components and seek for most probable fault propagation sequences based on anomaly analysis. We evaluate the RCSF method based on real enterprise system data and compare it with some baseline methods. Experimental results show that our proposed method can effectively anchor the root causes of failures by providing a short list of most probable causes, and the performance is significantly improved compared to the baseline methods.

I. INTRODUCTION

In a large distributed enterprise system, a monitoring system is usually employed to monitor anomalies. When an anomaly alarm is triggered, system managers should handle it in time to avoid cascading failures. However, the increasing size and complexity of enterprise applications makes anomaly management highly challenging. Faults can arise from a large variety of root-causes. The monitoring systems may generate a large number of alarms every day, making it infeasible to analyze them manually.

In a component based enterprise system, anomaly detectors are commonly deployed to monitor application-level components (business components), but not on functional components, such as routing nodes and data nodes, due to high cost and complexity of deployment. However, the root cause of anomalies can be the failure of a functional component, whereas most root diagnosis solutions, such as [5] and [10], assume all components are monitored. In this paper, we

propose RCSF, a root cause analysis method which does not require full monitoring coverage of components. It performs anomaly analysis on functional components based on their performance logs and seeks most probable fault propagation sequence between components. Our method is easy to deploy on existing monitoring systems since performance logs are commonly in place for functional components. Our real-data experimental results show that our method can provide an effective short list of most probable root cause sequences, which can help system managers locate real root causes efficiently. The comparison results with some baseline methods also demonstrate a significant improvement on performance. To the best of our knowledge, this paper is the first in the literature which provides root cause analysis solutions for enterprise systems with anomaly monitoring on application level components only.

The rest of the paper is organized as follows. In Section 2, we give an overview of related work in the literature. Section 3 give a description of the fault detection and root cause identification problem we are going to solve. The methodology are described in Section 4, followed by experimental results in Section 5. In the last Section 6, we conclude our work and state possible future work.

II. RELATED WORK

Root cause analyzing (RCA) and fault detection have been well studied in computing system domain. Several techniques, such as ping [4], heartbeat [1], and HTTP error code monitoring [12], are frequently used to check whether a physical machine or an application is alive or not. There are also many statistical learning methods designed to detect application-level service failures in the system. For example, the work in [7] models the web-based system into a weighted graph and applies graph mining techniques to monitor the graph sequences for failure detection. However, this approach can only detect anomalies at the application level but cannot locate the root causes of failures. The paper [5] presents the concepts of causal and temporal correlation within a single conceptual framework. However, these traditional rule-based approaches

are not suitable in this case since the signals of faults are not visible in the systems. Magpie [3] uses stochastic context free grammar to model and monitor requests' control flow across multiple machines to detect component failures and localize performance bottlenecks. The work in [8] uses time series analysis to model the dependency between system variables for fault detection. However, in modern large distributed systems, the asynchronization problem from different system components compromises the effectiveness of the time series approach. The PeerWatch [9], a fault detection and localization tool, models the correlation between multiple instances based on a statistical technique, called the canonical correlation analysis (CCA), and utilizes the correlated components to cross check the status of each application instance. However, PeerWatch works on the virtualized consolidation systems, which cannot be applied to the existing component-based distributed systems. The paper [6] proposes a method that focuses on offline event pattern analysis. In our situation, we need an online solution. In addition, their solution does not solve the same problem. The patterns of root-cause for the burst of anomalies may be different in each time so we cannot use the historical correlation knowledge. Instead we depend on the pattern frequency.

III. PROBLEM DEFINITION

In this section, we first describe the background and the motivation of the problem. We then present the problem by describing its basic assumptions.

A. Background and Motivation

In a typical enterprise system, such as the one in enterprise T¹, the number of functional components in the system is in the scale of thousands. However, the number of functional components related to a specific business component is typically in the scale of dozens. Although the number of functional components involved in an alarm may be small, the number of possible invocation sequences that triggered the alarm can be enormous due to the complex invocation relationship among components. We call the graph *TCG* (Transaction Call Graph). We can construct TCG using discovery engine such as iQuilt [14]. For example, as shown in Figure 1, functional components $F1, F2, F4, F5, F6, F7$ have invocation relationship with $B1$, and there are 10 possible invocation sequences including $(\{F1\}, \{B1\})$, $(\{F2\}, \{B1\})$, $(\{F4\}, \{F1\}, \{B1\})$, and $(\{F6\}, \{F4\}, \{F1\}, \{B1\})$, where the first element denotes the root cause of alarms. Our goal is to find the most probable fault propagation sequences and the root causes of alarms. An automatic root cause analysis system that effectively selects a short list of the most probable fault propagation sequences and root causes can play an important role in assisting system managers improve their work efficiency on root cause analysis.

¹We obscured the company name according to the submission guideline

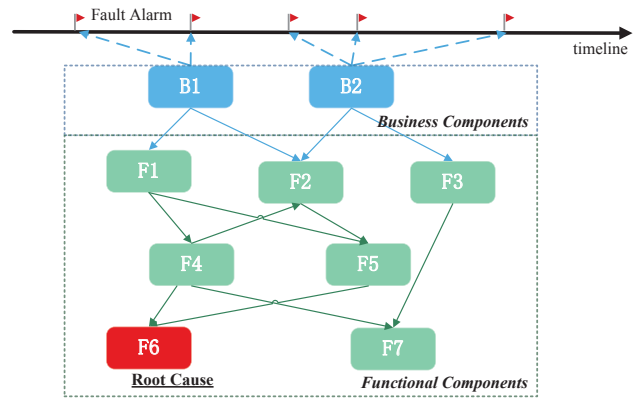


Fig. 1. Transaction Call Graph example

B. System Characters and Basic Assumptions

Designing a monitoring system for a large-scale distributed data center is a challenging task due to its scale and complexity. A single fault can cause multiple business components failure and deliver a large number of alarms to system managers. For example, as we observed in the enterprise T's Monitoring System, millions of alarms are generated every day due to various root causes. To provide the system administrators a short list of most probable root causes and fault propagation sequences, we designed RCSF system to automate the entire process. Our solution is designed for enterprise systems with the following properties:

- 1) It is a large-scale distributed enterprise system, where the monitoring sensors are deployed on application-level components, but not on functional components.
- 2) One root fault from a functional component may impact many application-level components and lead to a large number of alarms from different monitoring sensors.
- 3) Alarms resulting from one root cause appear within a short period of time, e.g. in an hour or two.
- 4) System administrator can verify the root causes of alarms given a short list of probable fault propagation sequences.

Our proposed solution also utilizes some additional properties that the enterprise system of company T possesses, which are listed as follows:

- 1) Fault alarms are generated by the application-level business components only. Since only the application-level components are monitored.
- 2) Both the business components and the functional components have the same performance patterns during each daily cycle.
- 3) If some faults occur on a functional component, its performance states show some anomalies correspondingly.

IV. METHODOLOGY

In this section, we describe the three major steps we use to analyze the root cause of faults.

A. Step 1: Anomalous functional components analysis

When a large number of anomaly alarms appear during a small time interval, the root cause may be an anomaly of a functional component which are related to the alarm generating business components. The related functional components can be retrieved using TCG. For example, as shown in Figure 1, F1~F7 are the functional components related to B1, B2. Our particular interest is to find components which have had anomaly before alarms appear by comparing their performance states with their historical performance states. We then rate the level of anomaly of each component.

In the enterprise environment, the performance state of a component consists of a set of sampling values. In our case, the sampling time interval is set to be 1 minute. The sampling target is the Key Performance Indicator (KPI) of a component.

T : denotes a collection of sampled points during a time-cycle.

T' : represents a collection of sampled points in the investigated time period. Our assumption is that all related alarms appear during a short time window after an anomaly occurs.

P_0 : is the collection of performance state samples in T' .

In a real enterprise context, a performance state is unlikely to be exactly identical to its historical values, since the value of p_0^k may change overtime. However, we can consider it normal if p_0^k remains in a range around the previous state. In our solution, we do not set a fixed range value to determine whether a component is anomalous or not, instead we assign the component a weight based on its distance to its expected normal state. If the component appears normal, it can be pruned in Step 2.

More specifically, let p_i^k denote the value of performance counter of a component at time moment k in the time-cycle i , where $t_k \in T$ and $i \in H$. We let P_H^k denote the set of performance samples in the investigated time period of a consecutive cycles sets. Then we have:

$$P_H^k = \{p_i^k | k \in [m, n], i \in H\} \quad (1)$$

We define the baseline of a performance counter at time t_k to be:

$$\overline{P^k} = \frac{\sum_H P_H^k - \max_H(P_H^k) - \min_H(P_H^k)}{H - 2} \quad (2)$$

Our purpose of subtracting the max and min value of P_H^k from the summation is to exclude the impact from performance "churn" in the system.

We identify a component to be anomalous if its value of p_0^k is distinct from the baseline $\overline{P^k}$. The anomalous level of a component can be computed as follows:

First, we measure their offset value Δ_k at time k by comparing their performance status with their historical baselines. Such that,

$$\Delta_k = \left\lceil \left| \frac{\overline{P^k} - p_0^k}{\overline{P^k}} \right| * 10 \right\rceil \quad (3)$$

We define k to be an anomalous point if $\Delta_k > \theta$, where θ is the system defined anomaly threshold parameter.

Second, we compute the average of all anomalous offsets, such that,

$$W_F^B = \frac{\sum_k \{\Delta_k | \Delta_k > \theta\}}{M} \quad (4)$$

where $M = \sum_{\Delta_k > \theta} 1$ is the total number of anomalous points.

We use W_F^B to denote the weight of the functional component F which has a revocational relationship with business component B . We are aware that the anomalous components we get may contain false positives, and we will discuss it in the next step.

B. Step 2: Fault propagation sequences analysis

The goal of this step is to exhaustively search for all invocation sequences which contain anomalous components as a propagation node. For example, in Figure 1, if B1 is the alarming component and F2 and F6 are anomalous components we get from Step 1, then there are 6 possible fault propagation sequences such as ($\{F2\}, \{B1\}$), ($\{F2\}, \{F4\}, \{F1\}, \{B1\}$), and ($\{F6\}, \{F4\}, \{F1\}, \{B1\}$). To avoid excessive number of invocation sequences, we further prune the components which are less likely to be anomalous. We generate candidate sequences using the following steps:

- *Pruning*: We prune the components which have the weight W_F^B under a specific threshold.
- *Fault propagation sequences generation*: We use an algorithm to calculate the propagation sequences that start with the anomalous components and end with the business components.

1) *Pruning*: The value of W_F^B can be used to represent the level of anomaly for component F .

We use a simple threshold τ_w to prune those components of weight W_F^B less than τ_w . The threshold is assigned by system managers based on their experience. The automatic configuration of τ_w will be studied in our future work.

2) *Fault propagation sequences generation*: A complete fault propagation sequence should start from a anomalous component and end with a business component. The algorithm of generating fault propagation sequences is described in Algorithm 1.

Input:

- $\mathbb{G} = \langle V(B), E(V(B), B) \rangle$ is the TCG of a specific business component B , where $V(B)$ is a set of functional components related to B , $E(V(B), B)$ are relationships between all components related to B .
- $fc = \{c | c \text{ is an anomalous functional component}\}$ is the collection of anomalous components.
- $bc = \{c | c \text{ is an alarming component during } T'\}$ is a collection of alarming business components we analyze.
- $getParents(c)$ returns all components that invoke component c .

Output:

- A list of probable fault propagation sequences.

Algorithm 1 Fault propagation sequences generation

```

Input : $\mathbb{G}, fc, bc$ 
Output : $a$  set of links
Set  $List < link > results \leftarrow \emptyset$ 
function  $getPathToHead(c_i)$ 
if  $c_i \in bc$  then
  attach  $c_i$  to link
  if exists a link  $\chi$  in  $results$  &  $link$  is a subsequence of  $\chi$  then
    return ;
  end if
  add  $link$  to  $results$ ;
  return ;
else if  $c_i \notin bc$  &  $c_i$  is business component then
  return ;
else
  for each component in  $getParent(c_i)$  do
    attach component to link;
     $getPathToHead(component)$ 
    remove component from link;
  end for
end if
mainEntry:
for each component in  $fc$  do
   $getPathToHead(component)$ 
end for
return  $results$ ;

```

C. Step 3: Frequent sequential pattern mining

The purpose of this step is to select *frequent sequence patterns* from many generated sequences. Frequent pattern mining (FPM) [11] is a branch of Data Mining. We assume that the root cause and subsequences of real fault propagation sequences appear more frequently than other sequences or components because they may be invoked by every alarming business components.

There are many mature algorithms of FPM, such as SPADE [15], PrefixSpan [13], SPAM [2], LAPIN, PLWAP and so on. In our methodology, we choose SPAM because of the following advantages [11]: 1) One-time database search; 2) Vertical projection of the database; 3) Better performance for large datasets.

V. EVALUATION

In this section, we evaluate our proposed root cause analysis system using real data from enterprise systems. Our evaluation results demonstrate that our method can generate an effective short list of most probable root cause sequences which provides the system managers guidance to identify root causes.

A. Experimental data

Our data is collected from one of the enterprise T's monitoring system. Enterprise T is one of the largest Internet service providers which provides mass media, entertainment, Internet mobile phone value-added services, and many other services. In this paper, all experiments were performed on a 3.4GHz Intel Core i7 PC machine with 4 gigabyte main memory, running on Java platform with windows 7.

Our experiment data was collected from the data-gathering platform. We collected outburst anomaly alarms on June 1st,

2014 and June 30th, 2014. There were 4 business components, 21 monitoring endpoints (for different monitoring dimensions), 80 functional components involved in alarms burst on June 1st, 2014. There are in total 248 alarms of different severities including 132 warnings and 116 critical alarms. We use W(A) to represent the dataset of June 1st and W(B) to represent the dataset of June 30th. The two datasets consist of the business components' alarms and performance sampling of related functional components. The performance we mentioned here is the combination of main KPI of the functional components. All alarms have been analyzed by system managers, and we know the actual root causes of all anomaly alarms.

The first alarming dataset W(A) was caused by a server-attack and the second alarming dataset W(B) was caused by a traffic migration within one datacenter. Table I lists the number of application-level business components (BS), the total number of anomaly alarms (AA), the total number of functional components (FC), total number of invocation paths from functional components (CP), and the invocation complexity of each dataset respectively. The invocation complexity is the average number of invocation paths per functional component. The larger the value, the more sequences we can generate from Step 2.

TABLE I

	BS	AA	FC	CP	Invocation complexity
W(A)	4	248	80	410	5.1
W(B)	3	179	71	425	6

We can see from Table I that, although the number of business components is few, the number of functional components is much larger. The number of call paths reaches four hundreds because of the complex invocation relationship among components. Since the probable fault propagation sequences are the combinations of different invocation paths, there can be thousands of sequences generated by Step 2 using our methodology in the real monitoring system.

B. Experiment setup

The input of the experiment include the datasets W(A) and W(B), TCG, and the performance state history of the related functional components. TCG is generated by system and stored in a two-dimension array.

In Step 1, we use TCG to find out all functional components related to the alarming business components, and get all the performance records of these components during a time period of two hours before alarming. The data is collected as tuple of $\langle \text{Time}, \text{Value} \rangle$ with a time interval of one minute. We collected 120 tuples for each functional component. Then we obtained the KPI of functional components during the same time period of 2 hours for the previous 14 days (H). After that we analyzed anomalous functional components with their weights through the algorithm provided in Step 1.

In Step 2, we firstly prune the anomalous components by their weight ($\tau_w = 15$), and then generate fault propagation sequences using TCG and the pruned components.

Finally, through frequent sequential pattern mining described in step 3, we get a list of suspected root causes and fault propagation sequences.

C. Experimental results

We use labeled datasets W(A) and W(B) from T's monitoring system to verify our methodology. In this section, we will analyze the results from three steps of our methodology correspondingly, discuss the relationships between functional components and propagation sequences, and analyze how the number of anomalous components affects the performance of our methodology.

TABLE II

	FC	RFC	GAC	RC in	IAC
W(A)	80	31	45	Y	23
W(B)	71	28	40	Y	21

1) *Anomalous functional components mining*: Through the algorithm we described in Step 1 of our methodology, we can get the list of probable anomalous components. Since the two datasets were analyzed by system managers, we can use those results as ground truth to evaluate the effectiveness of our method. Table II shows the results of the performance, including the total number of functional components (FC number), the real number of functional components that are involved in fault propagation sequence (RFC number), the number of anomalous components generated by our methodology (GAC number), whether the anomalous components we get contain the root cause (RC in), and the number of "invalid" anomalous components which are not included in real fault propagation sequences (IAC number).

2) *Fault propagation sequences analysis*: We generate the suspected fault propagation sequence by Step 2. The first step is to prune components with weight less than τ_w . τ_w is chosen to be a moderate value 15. This value can ensure keeping most of the components in real fault propagation sequence with a low false positive rate, and pruning most of the components which are likely to be non-critical. In our future work, we will study the automatic configuration of τ_w .

3) *Frequent sequential pattern mining*: In this step, we first mine the frequent sequential pattern from suspected sequences generated from Step 2 using SPAM as the FPM algorithm. The frequent sequential pattern can help system managers locate root causes and find subsequences of propagation sequence.

Let R(A) and R(B) be the experimental results of W(A) and W(B). Both R(A) and R(B) are a set of sequential patterns made up by one or more components, and all the sequential patterns are ranked by the frequency they appeared in the sequences exported from Step 2. We analyzed the top 5 ranked pattern from R(A) and R(B), finding that there are 3 and 2

TABLE III

	W(A)		W(B)	
	Sequences	Sup	Sequences	Sup
1	({F _A 1})	65	({F _B 2})	57
2	({F _A 2})	64	({F _B 1})	55
3	({F _A 3})	62	({F _B 1}, {F _B 2})	50
4	({F _A 1}, {F _A 2})	54	({F _B 3})	43
5	({F _A 1}, {F _A 3})	53	({F _B 2}, {F _B 3})	43

sequential patterns of the top 5 containing the actual root cause and all these sequential patterns are subsequences of the real fault propagation sequences. The system managers can locate the root cause by referring to the top n most supported sequential patterns produced by Step 3.

D. Comparison with Baseline Methods

In the previous subsections, we evaluated our methodology on T's monitoring system with real monitoring data. However, due to the particularity of this environment, we have not found any other root-cause method in the literature that can be adopted directly to this situation. In order to demonstrate the effectiveness of our proposed method, we compare our method with some baseline methods in this section. The baseline methods [10] we chose include two algorithms that locate root causes only, and one algorithm that finds both root cause and subsequences of propagation sequence. They can be described as follows:

1) *Random Selection (RS)*: A human without any domain knowledge may examine components in random order. We simulate this behavior by issuing random permutations.

2) *Components Weight Propensity (CWP)*: A natural way for a human to find root causes is to compare the metrics in the current and previous time windows and exam their difference. We refer to this offset value as the root cause score of each component.

3) *Components Weight Subsequences (CWS)*: To find subsequences of fault propagation sequence, we employ the method of components correlation using a dependency graph. For the root cause ranking score, we use every two components and find their propagation sequence in the dependency graph. Then, we rank the sequences by the sum of the root cause scores of these two components.

To compare our Root Cause Sequences Finding (RCSF) with the baseline methods, we require appropriate evaluation metrics. Since all the methods provide a rank of components or a rank of subsequences of fault propagation, we define the rank of a component i with respect to an alarm a as $r_a(i)$ and use the indicator variable $R_a(i)$ to represent whether component i is the root cause of an alarm a or not (that is, either 0 or 1). To quantify the performance of each algorithm on a set of alarms A, we use the following metrics:

Precision at top K (P@K) [10]: indicates the probability that each of the top K components given by each algorithm is actually the root causes of alarms. It is important that

TABLE IV

	RS	CWP	CWS	RCSF	Improvement compared to CWP	Improvement compared to CWS
P@1	0.042	0.686	0.753	0.987	43.9%	31.0%
P@3	0.047	0.748	0.693	0.932	24.6%	34.5%
P@5	0.043	0.763	0.742	0.914	19.8%	23.2%
MAP	0.045	0.749	0.712	0.969	29.4%	36.1%

the algorithm captures the final root cause at a small value of K , thereby resulting in lesser number of components to investigate. Here we use $K = 1, 3, 5$. More formally, it is defined as

$$P@K = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_{i: r_a(i) \leq K} R_a(i)}{\min(K, \sum_i R_a(i))} \quad (5)$$

Average Precision (MAP) [10]: quantifies the goodness of a give ranking result by putting more weight on the higher ranking results (where high rank component, which has lower $r_a(i)$, means that this component is more likely to be a root cause). For the number of components N , it can be formulated as follows:

$$MAP = \frac{1}{|A|} \sum_{a \in A} \sum_{1 \leq k \leq N} P@k \quad (6)$$

We evaluate RCSF and all the baseline methods on the throughput data that collected from one of the T's monitoring systems. In Table IV, we show the result of the comparison between our Root Cause Sequences Finding (RCSF) method and other baseline methods. We can see that the RCSF method outperforms the baseline methods by a large factor. The RS method demonstrates the weakest performance because this method only exams components in random manner and has no scientific justification. The CWP method shows a performance enhancement at a moderate pace with the increasing of K . However, in our real environment, it is important that the algorithm captures the final root cause at a small value of K . The CWS method shows an unbalanced performance, which may result in lower accuracy of the propagation sequence that we found. Comparatively, the result of RCSF shows a consistent performance and higher performance at P@1.

VI. CONCLUSION AND FUTURE WORK

In this paper we present a root cause sequence finding method (RCSF) for enterprise systems where only application-level business components have monitoring endpoints deployed, but not functional components. Our methodology is based on known dependency models about the whole system with no time constraint. Our methodology analyzes components' performance state and use three steps to provide a list of probable root causes of anomaly alarms. Our experiment results based on real data sets demonstrated that our

methodology can provide an list of effective root causes and subsequences of the fault propagation sequence which can help the system managers identify root causes in their daily work.

One of our future directions of our work is to have automatic configuration of the τ_w . The other direction is to locate the root causes more precisely and automatically by analyzing the frequent sequential patterns accompanied with the history knowledge of fault propagation sequences.

VII. ACKNOWLEDGEMENT

This research is supported by the National High Technology Research and Development Program of China (Grant No. 2012AA01A302), the National Natural Science Foundation of China (Grant No. 61361126011) and the Open Project Program of National Engineering Research Center for Science & Technology Resources Sharing Service(Beihang University).

REFERENCES

- [1] M. K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theoretical Computer Science*, 220(1):3–30, 1999.
- [2] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435. ACM, 2002.
- [3] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan. Magpie: Online modelling and performance-aware systems. In *HotOS*, pages 85–90, 2003.
- [4] W. Foundation. Ping. [http://en.wikipedia.org/wiki/Ping_\(networking_utility\)](http://en.wikipedia.org/wiki/Ping_(networking_utility)).
- [5] M. Hasan, B. Sugla, and R. Viswanathan. A conceptual framework for network management event correlation and filtering systems. In *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on*, pages 233–246. IEEE, 1999.
- [6] J. L. Hellerstein, S. Ma, and C.-S. Perng. Discovering actionable patterns in event data. *IBM Systems Journal*, 41(3):475–493, 2002.
- [7] T. Idé and H. Kashima. Eigenspace-based anomaly detection in computer systems. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 440–449. ACM, 2004.
- [8] G. Jiang, H. Chen, and K. Yoshihira. Discovering likely invariants of distributed transaction systems for autonomic system management. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 199–208. IEEE, 2006.
- [9] H. Kang, H. Chen, and G. Jiang. Peerwatch: a fault detection and diagnosis tool for virtualized consolidation systems. In *Proceedings of the 7th international conference on Autonomic computing*, pages 119–128. ACM, 2010.
- [10] M. Kim, R. Sumbaly, and S. Shah. Root cause detection in a service-oriented architecture. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, pages 93–104, New York, NY, USA, 2013. ACM.
- [11] N. R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys (CSUR)*, 43(1):3, 2010.
- [12] E. Marcus and H. Stern. *Blueprints for high availability*. John Wiley & Sons, 2003.
- [13] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 0215–0215. IEEE Computer Society, 2001.
- [14] M. Sethi, A. Anand, D. Gangopadhyay, V. Reddy, and M. Gupta. An open framework for federating integrated management model of distributed it environment. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 803–806. IEEE, 2008.
- [15] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2):31–60, 2001.