Localizing Runtime Anomalies in Service-Oriented Systems

Qiang He, Xiaoyuan Xie, Yanchun Wang, Dayong Ye, Feifei Chen, Hai Jin, Yun Yang*

Abstract—In a distributed, dynamic and volatile operating environment, runtime anomalies occurring in service-oriented systems (SOSs) must be located and fixed in a timely manner in order to guarantee successful delivery of outcomes in response to user requests. Monitoring all component services constantly and inspecting the entire SOS upon a runtime anomaly are impractical due to excessive resource and time consumption required, especially in large-scale scenarios. We present a spectrum-based approach that goes through a five-phase process to quickly localize runtime anomalies occurring in SOSs based on end-to-end system delays. Upon runtime anomalies, our approach calculates the similarity coefficient for each basic component (BC) of the SOS to evaluate their suspiciousness of being faulty. Our approach also calculates the delay coefficients to evaluate each BC's contribution to the severity of the end-to-end system delays. Finally, the BCs are ranked by their similarity coefficient scores and delay coefficient scores to determine the order of them being inspected. Extensive experiments are conducted to evaluate the effectiveness and efficiency of the proposed approach. The results indicate that our approach significantly outperforms random inspection and the popular Ochiai-based inspection in localizing single and multiple runtime anomalies effectively. Thus, our approach can help save time and effort for localizing runtime anomalies occuring in SOSs.

Index Terms—Program analysis, quality of service, Web service

1 Introduction

THE service-oriented computing paradigm offers an effective way to build software systems [10] that are composed of services locally or remotely accessed by an execution engine (e.g., a BPEL engine [34]). In such a service-oriented system (SOS), the component services jointly offer the functionality of the SOS and collectively fulfil its users' quality requirements.

Built from loosely coupled component services offered by independent (and often distributed) providers, SOSs operate in environments where key characteristics of the component services, such as the Quality of Service (QoS) properties, tend to be volatile. At runtime, various anomalies may occur in an SOS, e.g., unexpected workload changes, errors in the component services and failures of data transmissions, and impact on the quality of the SOS, causing end-to-end system delays. In this context, how to manage the quality of an SOS by detecting and adapting to runtime anomalies has become an important research direction [7, 10].

Response time, among various QoS dimensions, is of particular significance in quality management for SOSs. Amazon found that every extra 100ms of latency cost them 1% in sales [43] and Google found an extra 500ms in search page generation time dropped traffic by 20% [28]. The increase in the number of time-constrained applications in the cloud, e.g., interactive and multimedia SOSs, is also driving the needs for response time management for SOSs [26]. Furthermore, the management of response time is the basis for the management of other QoS dimensions. On one hand, effective response time management promises better management of other QoS dimensions because many applications exhibit trade-offs between their response times and other QoS dimensions [30]. A video encoding application, for example, can often produce higher quality video if it is given more time to encode the video frames. On the other hand, the management of other QoS dimensions is tightly coupled with response time management. During execution, an SOS may need to be adapted to fix runtime anomalies. The adaptation itself takes time, and as a result, contributes to delaying the execution of the SOS. Thus, timely detection of runtime anomalies is significant to effective quality management for SOSs.

An intuitive solution for timely detection of runtime anomalies is to constantly monitor all the *basic components* (BCs) of an SOS, including its component services and the data transmission links (or transmissions in short) between the component services. In response to a detected anomaly, adaptation actions can be taken before performance degradation becomes noticeable by the users. However, monitoring itself may incur excessive costs [7], making it impractical to constantly monitor the entire SOS, especially in large-scale scenarios where the number of BCs of the SOS and the number of SOSs are big. To address this issue, we proposed CriMon in [16] for formu-

- *Qiang He, Yanchun Wang, Dayong Ye and Feifei Chen are with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia 3122. E-mail: {qhe, yanchunwang, dye, feifeichen}@swin.edu.au.*
- *Xiaoyuan Xie is with the State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China. E-mail: xxie@whu.edu.cn.*
- *Hai Jin is with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: hjin@hust.edu.cn.*
- *Yun yang is with the School of Computer Science and Technology, Anhui University, China and the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia 3122. E-mail: yyang@swin.edu.au.*

*Please note that all acknowledgments should be placed at the end of the paper, before the bibliography (**note that corresponding authorship is not noted in affiliation box, but in acknowledgment section**).*

lating cost-effective monitoring strategies that focus on the BCs on the critical path. However, what if an anomaly occurs in an unmonitored BC? When that happens, the unmonitored BCs must be inspected to pinpoint the anomaly. Unfortunately, a comprehensive inspection of all unmonitored BCs can be expensive and sometimes impractical due to two potential costs, i.e., *resource cost* and *system cost*. First, invocations of services may be charged if those services are owned and hosted by different organisations [50]. Even if the invocations are free, the negative impact on the SOS caused by the inspection of its BCs might further degrade its quality [7]. For example, monitoring sometimes involves sniffing network traffic and retrieving the logs of services' and SOSs' behaviours. As demonstrated in [46], those operations can result in up to 70% performance overhead, slowing down the entire SOS as a result. We identified that a quality inspection can cause as much as 40% performance overhead on a Web service under certain circumstances [21]. These issues should not exclude inspection of BCs as a promising way of detecting anomalies. Unmonitored BCs still need to be inspected to localize runtime anomalies. To achieve timely anomaly detection at minimal resource cost and system cost, the BCs that are more likely to be faulty must be prioritised in the inspection. This way, the anomalies occurring in an SOS can be quickly pinpointed so that they can be fixed in time to minimise the sytem delays perceived by users.

To tackle this challenge, we propose a novel approach that employs spectrum-based fault localization (SFL) technique [2] to localize runtime anomalies in SOSs. There are other fault localization techniques, e.g., program slicing [49], delta debugging [31] and model-based diagnosis [29]. The reason for our choice of SFL is that it is the most light-weight fault localization technique [40]. SFL instruments and executes a program, before it is released, with a test suite to gather program spectrum. Each program spectrum is an abstraction of an execution trace. For each test case in the test suite, the program spectrum records which program components (e.g., statements and branches) were executed and whether the test case passed or failed. Then, SFL builds a matrix that consists of two parts: the program spectrum and the binary error vector. By evaluating the *similarity* between the program spectrum and the binary error vector, a heuristic measure for each program component, which expresses the *suspiciousness* of the program component being buggy and responsible for the test-case failures. The program components can then be ordered in decreasing order of suspiciousness to provide a ranking of program components from most to least suspicious, which are provided to program developers as guidance in debugging.

Localizing runtime anomalies in an SOS is very different from localizing faults in a program written in traditional languages, such as C, Java or PHP. Anomalies in an SOS occur at runtime and must be fixed as soon as possible. Running a comprehensive test suite to localize the anomalies often results in massive calls to the component services of the SOS. It is time consuming, resource consuming, and thus impractical. Although the component services of an SOS are usually distributed, the calls to those component services are carried out and managed by a centralised execution engine, e.g., a BPEL engine. Thus, end-to-end system delays caused by runtime anomalies can be detected by the execution engine with marginal system overhead. In response to different user requests, different parts of the SOS are activated. Thus, an SOS can be represented by multiple execution scenarios [16].

Following the methodology of SFL, upon detected end-to-end system delays, our approach generates *system spectra* that record which BCs are covered by which execution scenarios, and a binary *delay vector* that indicates which execution scenarios are experiencing delays. Then, it diagnoses the differences in the system spectra for *normal* and *delayed* execution scenarios, where normal execution scenarios are the execution scenarios with no delays and delayed execution scenarios are the execution scenarios where delays are occurring. By evaluating the similarity between the system spectrum and the binary delay vector, a *similarity coefficient* score is calculated for each BC of the SOS as a heuristic measure that expresses the *suspiciousness* of the BC being responsible for the system delay. SFL has been acknowledged as an effective approach for localizing single program faults, but it performs less favourably when localizing multiple program faults [24]. This inherent limitation of SFL automatically transmits to its application in the context of SOSs. To handle concurrent runtime anomalies, we further analyse the differentiated end-to-end delays in different execution scenarios to calculate a *delay coefficient* score for each BC to expresses the BC's contribution to the severity of end-to-end system delays. Next, the BCs are sorted first in decreasing order of their similarity coefficient scores and then their delay coefficient scores. Finally, the results are used to pinpoint the anomaly or, if an exact pinpoint cannot be found, presented as guidance that can reduce the SOS administrator's effort in searching for the anomaly.

The contributions of this paper are as follows:

- We propose a spectrum-based approach for localizing runtime anomalies upon end-to-end system delays. This approach does not require constant monitoring or comprehensive inspection of all BCs of the SOS.
- We propose the calculation of similarity coefficient and delay coefficient, based on which BCs can be ranked to indicate their suspiciousness of being faulty. The ranking provides system administrators with guidelines for localizing runtime anomalies.
- Extensive and comprehensive experiments are conducted to evaluate our approach using a published real-world Web service dataset, which contains over 2500 real-world Web services. The evaluation shows that our approach significantly outperforms random inspection and traditional SFL. Using our approach, the time and efforts for localizing runtime anomalies occurring in SOSs can be significantly saved.

The rest of this paper is organised as follows: Section 2 analyses the requirements with a motivating example. Section 3 describes our anomaly localization approach. Section 4 presents the experimental results to demonstrate effectiveness and efficiency of our approach. Section 5 reviews related work. Section 6 concludes the paper and points out future work.

2 Motivating Example

This section presents an example SOS, namely OnlineLive, to motivate this research. As depicted in Fig. 1, this SOS offers an on-demand service to convert, subtitle and transmit live video streams. OnlineLive consists of 26 BCs. N_1, N_2, …, N_8 represent the component services and E_A, E_B, …, E_R represent the data transmissions between the component services. In response to a user request, the execution process of OnlineLive is as follows:

Step 1: N_1 splits the live media stream selected by the user into separate video and audio streams.

Step 2: The video and audio streams are processed in parallel. Specifically,

- For normal users, N_2 encodes 360p video stream and N_3 embeds advertisements into the video stream. For ad-free premium users, N_4 encodes 1080p video stream.
- N_5 generates the subtitle by performing speech recognition on the audio stream. Then, based on the user's preference or country/region, the subtitle is sent to either N_6 or N_7 to be translated into one of the two optional languages.

Step 3: N_8 produces a media stream by merging and synchronising the video stream, audio stream and translated subtitle.

Step 4: The media stream is transmitted to the user.

OnlineLive must process the media stream timely and continuously. Otherwise, the user will receive a jittering media stream. When anomalies occur and cause delays to OnlineLive, the BCs must be inspected to identify the faulty ones. Random inspection is a simple approach. However, we must prioritise the BCs that are more likely to be faulty in order to localize the anomalies rapidly. By doing so, adaptation actions can be taken to fix the anomaly in time to avoid or reduce the system delay caused by the anomalies.

The most intuitive solution for timely detection of runtime anomalies is to constantly monitor all the BCs of OnlineLive. Another solution is to comprehensively inspect the status and quality of all BCs every time a system delay occurs. As discussed in Section 1, both solutions are expensive in terms of time and resource consumption, and thus are often impractical. Another approach is to employ end-to-end quality information of OnlineLive for

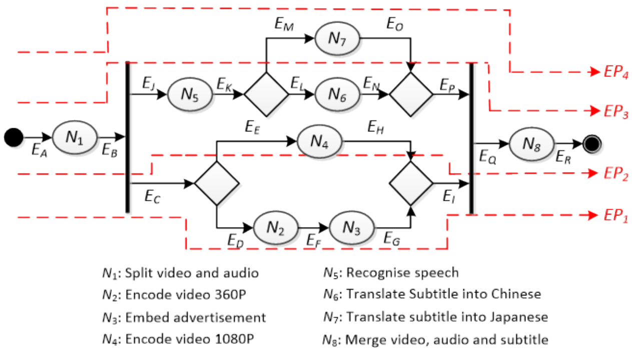
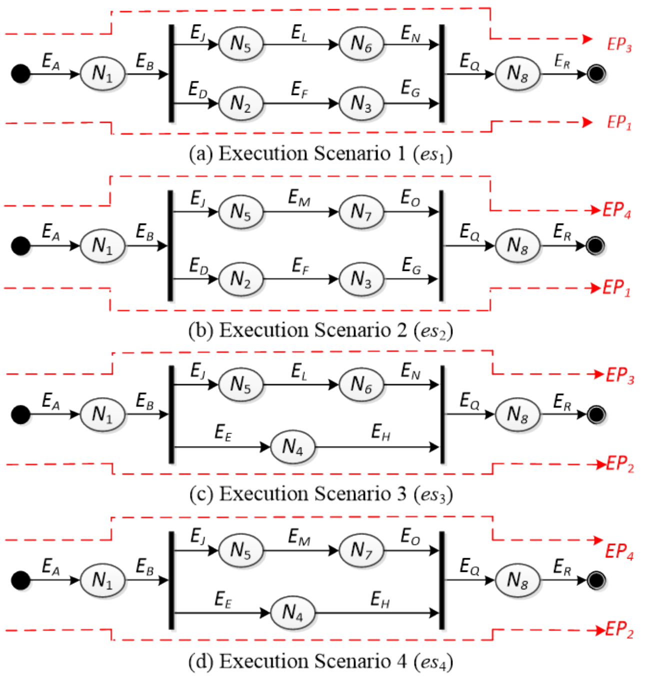


Fig. 1. Process of OnlineLive.



E_A N_1 E_B E_J N_5 E_L N_6 E_N E_Q N_8 E_R EP_3
E_D N_2 E_F N_3 E_G EP_1
(a) Execution Scenario 1 (es_1)
E_A N_1 E_B E_J N_5 E_M N_7 E_O E_Q N_8 E_R EP_4
E_D N_2 E_F N_3 E_G EP_1
(b) Execution Scenario 2 (es_2)
E_A N_1 E_B E_J N_5 E_L N_6 E_N E_Q N_8 E_R EP_3
E_E N_4 E_H EP_2
(c) Execution Scenario 3 (es_3)
E_A N_1 E_B E_J N_5 E_M N_7 E_O E_Q N_8 E_R EP_4
E_E N_4 E_H EP_2
(d) Execution Scenario 4 (es_4)

Fig. 2. Execution scenarios of OnlineLive.

localizing the occurring anomalies. Four execution scenarios can be identified from OnlineLive: es_1={EP_1, EP_3}, es_2={EP_1, EP_4}, es_3={EP_2, EP_3} and es_4={EP_2, EP_4}, as presented in Fig. 2. Which one will be executed in response to a user request is dependent on the runtime decisions made at the two branch structures. Assume that the constraint for the response time of OnlineLive is 3.0s but now it is taking OnlineLive 3.5s to process user requests, resulting in a system delay. System logs indicate that the current end-to-end response times of es_1, es_2 es_3 and es_4 are 3.3s, 2.4s, 3.5s and 2.5s respectively. Obviously, es_1 and es_3 have violated the constraint of response time, thus there must be at least one faulty BC in both of these two execution scenarios. By comparing these four execution scenarios, we can find that: E_L, N_6 and E_N are exclusively invoked by these two delayed execution scenarios; E_M, N_7 and E_O are only involved in the normal execution scenarios; while the rest BCs are included in both cases. Therefore, we can reasonably infer that E_L, N_6 and E_N are the most suspicious BCs that result in the system delay. Furthermore, multiple concurrent anomalies occurring in OnlineLive will cause a more severe end-to-end system delay, which is determined by the execution scenario with the maximum execution time among es_1, es_2 es_3 and es_4. If one execution scenario is experiencing an especially severe delay, the BCs that belong to that execution scenario, especially those that exclusively belong to that execution scenario, are more likely to be faulty. Those BCs must be prioritised in the inspection.

By analysing end-to-end quality information, our approach can help the system administrator localize the faulty BCs quickly by ranking the BCs of Onlive in decending order of their suspiciousness of being faulty.

3 Anomaly Localization

Our anomaly localization approach is designed as a five-phase process as shown in Fig. 3. In Phase 1, the loops in the SOS are peeled. In Phase 2, a set of execution scenarios are identified from the SOS. Then, in Phase 3, the system spectra are constructed. In Phase 4, similarity coefficients and delay coefficients are calculated for the BCs of the SOS. Finally, in Phase 5, the BCs are ranked based on their similarity coefficients and delay coefficients to suggest an order in which the BCs should be inspected. Details of these phases are presented in Section 3.1 to Section 3.5 respectively.

3.1 Phase 1: Loop Peeling

In this research, we use four types of basic compositional structures, i.e., *sequence*, *branch*, *loop* and *parallel* for representing and analysing SOS. These compositional structures are included in BPMN [35], and addressed by BPEL [34] - the de facto standard for specifying service-oriented business processes. They are also adopted in many other studies on SOS [5, 16, 50].

- **Sequence.** In a sequence structure, the BCs are executed one by one.
- **Branch.** In a branch structure, only one branch is selected for execution. For a set of branches $\{b_1, \ldots, b_n\}$, the execution probability distribution $\{p(b_1), \ldots, p(b_n)\}$, $(0 \leq p(b_i) \leq 1, \sum_{i=1}^{n} p(b_i) = 1.0)$ is specified, where $p(b_i)$, $i=1, \cdots, n$, is the probability that the ith branch is selected for execution.
- **Loop.** In a loop structure, the loop is executed for n $(0 \leq n \leq MNI)$ times. For a loop, the probability distribution $\{p_0, \ldots, p_{MNI}\}$, $(0 \leq p_i \leq 1, \sum_{i=0}^{MNI} p_i = 1.0)$ is specified, where p_i, $i=0, \ldots, MNI$, is the probability that the loop iterates for i times and MNI is the expected maximum number of iterations for the loop.
- **Parallel.** In a parallel structure, all the branches are executed at the same time.

The probabilities, $p(b_i)$, p_i and the maximum number of iterations can be evaluated based on the past executions of the SOS or can be specified by the developer [5]. We assume that for a loop, the *MNI* can be determined or estimated. Otherwise, without an upper bound for the number of iterations, the execution times of the execution paths that contain the loop cannot be calculated since the loop may iterate infinitely.

We represent service compositions using UML activity diagrams, where the nodes represent component services and the edges represent data transmissions. Without losing generality, we assume that a service composition is characterised by only one entry point and one exit point, and only includes *structured loops* with only one entry point and one exit point. If a service composition includes loops, we peel the loops by representing loop iterations as a set of branches with corresponding execution probabilities [5]. Fig. 4 gives an example of peeling a loop structure (*MNI*=2) by transforming it into a branch structure that contains three branches b_1, b_2 and b_3, where p_0, p_1 and p_2 are the probabilities that b_1, b_2 and b_3 are selected for execution respectively. (Note that the first branch b_1 is selected if the loop iterates for 0 times, i.e., corresponding to p_0).

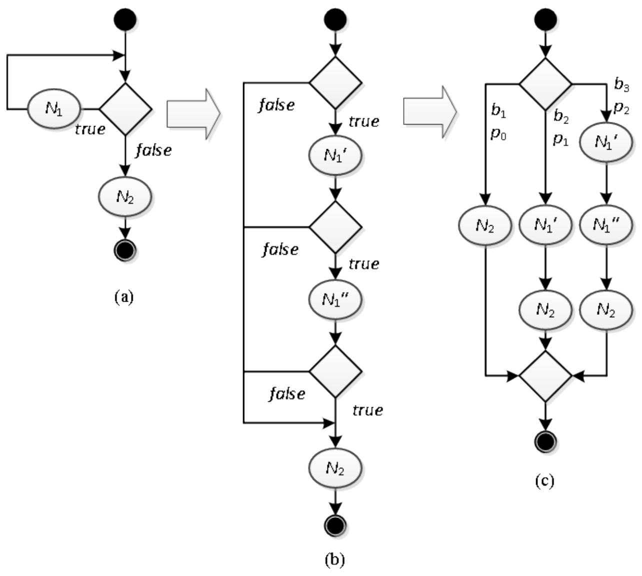


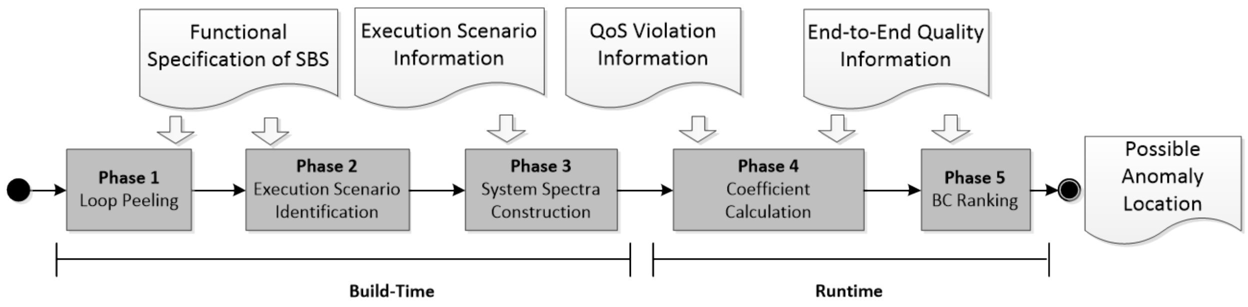
Fig. 4. Loop peeling process.

3.2 Phase 2: Execution Scenario Identification

In a service composition where branches or loops are involved, different execution paths may be selected for execution. Thus, multiple *execution scenarios* can be identified from the service composition. These execution scenarios do not contain branch or loop structures. As depicted in Fig. 2, four execution scenarios can be identified from OnlineLive: $es_1=\{EP_1, EP_3\}$, $es_2=\{EP_1, EP_4\}$ $es_3=\{EP_2, EP_3\}$ and $es_4=\{EP_2, EP_4\}$. The identified execution scenarios will be used for constructing the system spectra as described next in Section 3.3.

3.3 Phase 3: System Spectra Construction

Anomaly localization requires analysis of the differences in system spectra [42] for individual execution scenarios of the SOS. A system spectrum is a collection of execution traces that shows which BCs were included during an execution of the SOS. In this research, a system spectrum contains a flag for each BC of the SOS to indicate whether


Functional Specification of SBS
Execution Scenario Information
QoS Violation Information
End-to-End Quality Information
Phase 1 Loop Peeling
Phase 2 Execution Scenario Identification
Phase 3 System Spectra Construction
Phase 4 Coefficient Calculation
Phase 5 BC Ranking
Possible Anomaly Location
Build-Time
Runtime
Fig. 3. Anomaly localization approach.

$$\begin{array}{ccc} & m \text{ BCs} & \text{Delay Vector} \\ n \text{ execution scenario} & \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{bmatrix} & \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \end{array}$$

Fig. 5. System spectra and delay vector.

or not that BC is included in a particular execution scenario of the SOS. Take OnlineLive as an example. There are four system spectra, one for each execution scenario, as presented by the four dark grey columns in Table 1.

The system spectra of n execution scenarios constitute an $n \times m$ binary matrix, where m is the number of BCs in an SOS, as presented in Fig. 5. Given $i \in [1,\dots,n]$ and $j \in [1,\dots,m]$, c_{ij}=1 indicates that the j^{th} BC is included in the i^{th} execution scenario. The four dark grey columns in Table 1 constitute the system spectra of OnlineLive. Each row in the matrix is a system spectrum that indicates whether a BC is part of an execution scenario. For example, the first row, (1, 1, 1, 1), indicates that N_1 is part of all four execution scenarios, es_1, es_2, es_3 and es_4. The second row, (1, 1, 0, 0) indicates that N_2 is part of es_1 and es_2, but not es_3 and es_4. The system spectra will be used in the similarity coefficient calculation for anomaly localization discussed next in Section 3.4.

3.4 Phase 4: Coefficient Calculation

This section presents the calculation of two coefficients, similarity coefficient and delay coefficient. They indicate the suspiciouseness of BCs being faulty and are used in Phase 5 for ranking the BCs for inspection.

3.4.1 Similarity Coefficient

A similarity coefficient score is calculated based on a BC's involvement in all the normal and delayed execution scenarios. It indicates the suspiciousness of a BC being faulty. It also approximates the reliability of a BC [41], where a high similarity coefficient score implies low reliability.

In Fig. 5, the binary delay vector that contains n flags is used to represent which execution scenarios are experiencing a delay in processing user requests. Given $i \in [1,\dots,n]$, v_i=1 indicates that es_i is experiencing a delay, and 0 otherwise. To complete the delay vector, the time consumed by each execution scenario in processing user requests needs to be logged. It can be obtained from the execution engine of the SOS by calculating the time difference between the arrival of user requests and the delivery of corresponding outcomes. This is feasible because the execution engine that invokes the distributed services is centrally managed by the SOS provider [38]. Each request will traverse one execution scenario, depending on the runtime decisions made in the branch structures of the SOS. Which execution scenario each user request traverses also needs to be recorded to complete the delay vector depicted in Fig. 5. This can be achieved by recording the runtime decisions made at the branches by the execution engine of the SOS. Now assume that an anomaly is occurring in N_6, slowing it down and violating the constraint for the response time of OnlineLive. Because N_6

TABLE 1
SYSTEM SPECTRA, SIMILARITY COEFFICIENT AND BC RANKS FOR ONLINELIVE
(ANOMALY OCCURING IN N_6)

Basic Component	Execution Scenario				n_{11}	n_{10}	n_{01}	n_{00}	Similarity Coefficient			Ranking (by c_O)
	es_1	es_2	es_3	es_4					c_J	c_T	c_O	
N_1	1	1	1	1	2	2	0	0	0.50	0.50	0.71	11
N_2	1	1	0	0	1	1	1	1	0.33	0.50	0.50	19
N_3	1	1	0	0	1	1	1	1	0.33	0.50	0.50	19
N_4	0	0	1	1	1	1	1	1	0.33	0.50	0.50	19
N_5	1	1	1	1	2	2	0	0	0.50	0.50	0.71	11
N_6	1	0	1	0	2	0	0	2	1.00	1.00	1.00	3
N_7	0	1	0	1	0	2	2	0	0.00	0.00	0.00	22
N_8	1	1	1	1	2	2	0	0	0.50	0.50	0.71	11
E_A	1	1	1	1	2	2	0	0	0.50	0.50	0.71	11
E_B	1	1	1	1	2	2	0	0	0.50	0.50	0.71	11
E_D	1	1	0	0	1	1	1	1	0.33	0.50	0.50	19
E_E	0	0	1	1	1	1	1	1	0.33	0.50	0.50	19
E_F	1	1	0	0	1	1	1	1	0.33	0.50	0.50	19
E_G	1	1	0	0	1	1	1	1	0.33	0.50	0.50	19
E_H	0	0	1	1	1	1	1	1	0.33	0.50	0.50	19
E_J	1	1	1	1	2	2	0	0	0.50	0.50	0.71	11
E_L	1	0	1	0	2	0	0	2	1.00	1.00	1.00	3
E_M	0	1	0	1	0	2	2	0	0.00	0.00	0.00	22
E_N	1	0	1	0	2	0	0	2	1.00	1.00	1.00	3
E_O	0	1	0	1	0	2	2	0	0.00	0.00	0.00	22
E_Q	1	1	1	1	2	2	0	0	0.50	0.50	0.71	11
E_R	1	1	1	1	2	2	0	0	0.50	0.50	0.71	11
Delay	1	0	1	0								

is included in EP_3, the system delay is only observed in es_1 and es_3. The sytem delay vector will be: (1, 0, 1, 0). Table 1 presents the system spectra constructed for OnlineLive.

The similarity between the binary delay vector and a system spectrum indicates the suspiciousness of the corresponding BC being responsible for the occurring system delay. In Table 1, only the system spectra corresponding to N_6, E_L and E_N are consistent with the delay vector – they are all (1, 0, 1, 0). Thus, N_6, E_L and E_N can be identified as the most suspicious BCs and should be inspected with the highest priority.

For large-scale applications, the system spectra and the delay vectors can be remarkably large and complex. The suspicious BCs are those whose corresponding system spectra are most statistically similar to the delay vector. In order to statistically analyse the similarity between the system spectra and the delay vector, we now extract the features of the raw data in the system spectra and construct a *feature matrix* by comparing the system spectra with the delay vector. The feature matrix has four columns for representing four features:

$$n_{pq}(S_j) = |\{i \mid x_{ij} = p \wedge v_i = q\}|,\ p,q \in \{0,1\} \quad (1)$$

where x_{ij}=1 indicates that BC S_j is included in es_i, and 0 otherwise, v_i=1 indicates that a delay is occurring in es_i (i.e., a delayed execution scenario) and 0 (a normal execution scenario) otherwise. $n_{10}(S_j)$ and $n_{11}(S_j)$ are the numbers of normal and delayed execution scenarios respectively where S_j is involved. $n_{00}(S_j)$ and $n_{01}(S_j)$ are the numbers of normal and delayed execution scenarios respectively where S_j is NOT involved.

Now the statistical similarity between each of the system spectrum and the delay vector can be quantified by calculating the similarity coefficients of the BCs. Calculated in different ways, many similarity coefficients exist, e.g., Jaccard coefficient c_J, used by the Pinpoint tool [12], the coefficient c_T used in the Tarantula fault localization tool [25] and the Ochiai coefficient c_O often used in the molecular biology domain [14]:

$$c_J(S_j) = \frac{n_{11}(S_j)}{n_{11}(S_j) + n_{01}(S_j) + n_{10}(S_j)} \quad (2)$$

$$c_T(S_j) = \begin{cases} 1.00 & \text{if } n_{10}(S_j) + n_{00}(S_j) = 0 \\ \dfrac{\dfrac{n_{11}(S_j)}{n_{11}(S_j) + n_{01}(S_j)}}{\dfrac{n_{11}(S_j)}{n_{11}(S_j) + n_{01}(S_j)} + \dfrac{n_{10}(S_j)}{n_{10}(S_j) + n_{00}(S_j)}} & \text{otherwise} \end{cases} \quad (3)$$

$$c_O(S_j) = \frac{n_{11}(S_j)}{\sqrt{(n_{11}(S_j) + n_{01}(S_j)) \times (n_{11}(S_j) + n_{10}(S_j))}} \quad (4)$$

where S_j is a BC. A higher similarity coefficient indicates higher suspiciousness of a BC being faulty.

3.4.2 Delay Coefficient

In a volatile operating environment, multiple anomalies may occur at the same time to different BCs of an SOS, especially for large-scale SOSs that consist of many BCs. Multiple concurrent anomalies usually cause more severe delays than a single anomaly. As discussed in Section 2, the BCs in an execution scenario where a more severe delay is occurring are more suspicious of being faulty. However, it is not reflected by the Jaccard coefficient, Tarantula coefficient and Ochiai coefficient because they take into account only whether there is a delay in an execution scenario. The binary delay vector does not indicate the severity of the delay occurring in each of the execution scenarios. When multiple anomalies occur at the same time, the probability is high that most, sometimes even all, of the execution scenarios will be delayed, leading to a delay vector with many 1s, even full 1s, i.e., (1, 1, …, 1). Such a delay vector will lower the variety of the feature vectors (the light grey rows) in the feature matrix, making it harder to use the similiarity coefficients to distinguish the BCs from each other.

Take Fig. 2 for example: suppose two anomalies are occurring, one to service N_5 and the other to service N_6. Table 2 presents the corresponding system spectra, similarity coefficients and BC ranks. As presented by the corresponding system spectra, N_5 belongs to all four execution scenarios and N_6 belongs to es_1 and es_3. N_5 causes a delay to all four execution scenarios, resulting in a delay vector (1, 1, 1, 1). Constructed based on the comparison between the system spectra and the all-1 delay vector, the feature matrix contains only (4, 0, 0, 0) and (2, 2, 0, 0). The consequence is low variety in the similarity coefficients and rankings for the BCs. As presented by the *Ranking (by c_o)* column, besides N_5, seven other BCs are ranked 8 as the most suspicious BCs. N_6 is ranked 21 together with all the other BCs. In the best case scenario, N_5 is pinpointed as a fault BC at the first inspection of all the BCs ranked 8 and N_6 is pinpointed as an abnormal BC at the second inspection. The total number of BCs that must be inspected to pinpoint N_5 and N_6 is 2. In the worse case scenario, the number is 21. On average, it is 11.5, getting close to random inspection, which needs to inspect an average of 16.18 BCs to pinpoint both N_5 and N_6. Apparently, the traditional SFL techniques based on Jaccard coefficient, Tarantual conefficient and Ochiai coefficient cannot handle concurrent anomalies effectively.

In order to address this issue, we need to take into account the severity of the delays caused to each execution scenario. In Fig. 2, concurrent anomalies occurring in N_5 and N_6 will cause more severe delays to es_1 and es_3 compared to es_2 and es_4 because es_1 and es_3 contain both N_5 and N_6 while es_2 and es_4 contain only N_5. We calculate the *delay coefficient*, denoted by c_d, to further indicate the suspiciousness of a BC based on the severity of the delays caused by anomalies to the execution scenarios. Given a set of $res_j(es_i)$, j=1, …, h, logged as the response time of execution scenario es_i for processing n user requests after the systeme delay is detected, we calculate the standard deviation of $res_j(es_i)$ (j=1, …, h) from its normal value $\overline{res(es_i)}$:

$$SD(res(es_i)) = \sqrt{\frac{1}{h}\sum_{j=1}^{h}(res_j(es_i) - \overline{res(es_i)})^2} \quad (5)$$

We measure the severity of a delay by measuring the

TABLE 2
SYSTEM SPECTRA, SIMILARITY COEFFICIENT AND BC RANKS FOR ONLINELIVE
(ANOMALIES OCCURING IN N_5 AND N_6)

Basic Component	Execution Scenario				n_{11}	n_{10}	n_{01}	n_{00}	Similarity Coefficient			Ranking (by c_O)	Delay Coefficient c_D	Ranking (by c_D)
	es_1	es_2	es_3	es_4					c_J	c_T	c_O			
N_1	1	1	1	1	4	0	0	0	1	1	1	8	0.47	8
N_2	1	1	0	0	2	0	2	0	0.5	0.5	0.71	22	0.55	16
N_3	1	1	0	0	2	0	2	0	0.5	0.5	0.71	22	0.55	16
N_4	0	0	1	1	2	0	2	0	0.5	0.5	0.71	22	0.39	19
N_5	1	1	1	1	4	0	0	0	1	1	1	8	0.47	8
N_6	1	0	1	0	2	0	2	0	0.5	0.5	0.71	22	0.89	11
N_7	0	1	0	1	2	0	2	0	0.5	0.5	0.71	22	0.05	22
N_8	1	1	1	1	4	0	0	0	1	1	1	8	0.47	8
E_A	1	1	1	1	4	0	0	0	1	1	1	8	0.47	8
E_B	1	1	1	1	4	0	0	0	1	1	1	8	0.47	8
E_D	1	1	0	0	2	0	2	0	0.5	0.5	0.71	22	0.55	16
E_E	0	0	1	1	2	0	2	0	0.5	0.5	0.71	22	0.39	19
E_F	1	1	0	0	2	0	2	0	0.5	0.5	0.71	22	0.55	16
E_G	1	1	0	0	2	0	2	0	0.5	0.5	0.71	22	0.55	16
E_H	0	0	1	1	2	0	2	0	0.5	0.5	0.71	22	0.39	18
E_J	1	1	1	1	4	0	0	0	1	1	1	8	0.47	8
E_L	1	0	1	0	2	0	2	0	0.5	0.5	0.71	22	0.89	11
E_M	0	1	0	1	2	0	2	0	0.5	0.5	0.71	22	0.05	22
E_N	1	0	1	0	2	0	2	0	0.5	0.5	0.71	22	0.89	11
E_O	0	1	0	1	2	0	2	0	0.5	0.5	0.71	22	0.05	22
E_Q	1	1	1	1	4	0	0	0	1	1	1	8	0.47	8
E_R	1	1	1	1	4	0	0	0	1	1	1	8	0.47	8
Delay	1	1	1	1										

standard deviation of the response times of execution scenarios. To compare the severity of delays caused by the anomalies to different execution scenarios es_i (i=1, …, m) on the same scale independently of their original response times, we calculate the coefficient of variation for each es_i:

$$c_v(es_i)=\frac{SD(res(es_i))}{\overline{res(es_i)}}=\frac{\sqrt{\frac{1}{h}\sum_{j=1}^{h}(res_j(es_i)-\overline{res(es_i)})^2}}{\overline{res(es_i)}} \quad (6)$$

Next, the min-max normalisation technique, which has been widely used in a number of approaches for QoS evaluation [5, 20, 48], is employed to normalise the coefficients of variation as follows:

$$\hat{c}_v(es_i)=\begin{cases}\dfrac{c_v(es_i)-c_v^{min}(es_i)}{c_v^{max}(es_i)-c_v^{min}(es_i)} & \text{if } c_v^{max}(es_i)\neq c_v^{min}(es_i)\\ 1 & \text{if } c_v^{max}(es_i)=c_v^{min}(es_i)\end{cases} \quad (7)$$

where $c_v^{max}(es_i)$ and $c_v^{min}(es_i)$ are the maximum and minimum values in $c_v(es_i)$, i=1, …, m. Suppose two anomalies are occurring, one to N_5 and the other to N_6 in Fig. 2. Table 3 presents the calculation of the normalised coefficients of variation for es_1, es_2, es_3 and es_4.

The BCs that belong to an execution scenario with a severe delay (indicated by a high c_v) should be prioritised in the inspection. However, a BC can belong to multiple execution scenarios. Thus, we average the coefficients of variation for all the execution scenarios that a BC S_j belongs to to calculate the delay coefficient of the BC:

$$c_D(S_j)=\frac{\sum_{i=1}^{n}x_{ij}\cdot\hat{c}_v(es_i)}{\sum_{i=1}^{n}x_{ij}} \quad (8)$$

where x_{ij} is 1 if S_j belongs to es_i and 0 otherwise.

The *Delay Coefficient* c_D column in Table 2 presents the delay coefficients of the BCs of OnlineLive calculated based on Table 3.

TABLE 3
DELAY COEFFICIENT CALCULATION
(ANOMALIES OCCURING IN N_5 AND N_6)

	es_1	es_2	es_3	es_4
$\overline{res(es)}$	2.30	2.40	2.50	2.50
$res_1(es)$	4.82	3.46	4.77	3.46
$res_2(es)$	4.85	3.39	4.82	3.41
$res_3(es)$	4.75	3.41	4.83	3.39
$res_4(es)$	4.73	3.50	4.76	3.32
$SD(res(es))$	2.49	1.04	2.30	0.90
$c_v(es)$	1.08	0.43	0.92	0.36
$\hat{c}_v(es)$	1.00	0.10	0.77	0.00

3.5 Phase 5: BC Ranking

Based on the similarity coefficient scores, the BCs can be ranked from high to low, where the highest indicates the most suspicious. When multiple BCs share a same coefficient score, we use the approach reported in [44]: all tied BCs get the greatest ranking number. The *Ranking* (*by* c_O) columns in Table 1 and Table 2 present the rankings of all BCs in OnlineLive in the cases of one anomaly (occurring in N_6) and two anomalies (occurring in N_5 and N_6) respectively. BCs can also be ranked by their delay coefficient scores. Based on their similarity coefficients and delay coefficients, we can rank BCs in different ways. One option is to first rank them by their similarity coefficient scores and then rank the ones sharing the same similarity coefficient scores by their delay coefficient scores. Table 2 presents the BCs of OnlineLive ranked in this way. 14 BCs, are ranked 22 by their similarity coefficient scores, as indicated by column *Ranking* (*by* c_O). They are further ranked 11, 16, 19 and 22 by their delay coefficient scores, as indicated by column *Ranking* (*by* c_D). The ranking provides system administrator with an order in which the BCs are inspected. For OnlineLive, starting from the BC with the highest ranking, 2 BCs in the best case scenario and 11 BCs in the worst case scenario need to be inspected to pinpoint both faulty BCs, i.e., N_5 and N_6. On average, 6.5 BCs need to be inspected to pinpoint both N_5 and N_6, requiring much less BCs to be inspected to localize all faulty BCs compared to random inspection which needs to inspect an average of 16.18 BCs.

The BCs can also be ranked in other ways. For example, they can be ranked first by their delay coefficient scores then by their similarity coefficient scores. The similarity coefficient and delay coefficient can also be combined into one coefficient for ranking the BCs. For example, BCs can be ranked by the average of their similarity coefficient scores and delay coefficient scores, i.e., $(c_O+c_D)/2$. In Section 4, we will experimentally evaluate the three ranking methods discussed above.

4 Experiments

We conducted a range of experiments in a simulated volatile environment to evaluate our anomaly localization approach. Section 4.1 describes the setup of the experiments and Section 4.2 presents the experimental results.

4.1 Experimental Setup

For the evaluation, we have implemented the following inspection approaches:

- *Random inspection*. In response to any anomalies, this approach inspects the BCs randomly to localize the faulty BCs. It stops when all anomalies are localized. Inspected BCs are not reinspected. This inspection approach is the baseline for the evaluation.
- c_O *Ranking*. This inspection approach was proposed and evaluated by us in [18]. In response to runtime anomalies, this approach evaluates the suspiciousness of a BC by calculating the Ochiai-based similarity coefficients of the BCs. The BC with highest similarity efficient is inspected first. This approach also stops when all anomalies are localized. As discussed in Section 3.4, there are other coefficients that can be used to calculate the similarity of BCs, e.g., Jaccard and Tarantula. We adopt the Ochiai coefficient because it outperforms the Jaccard and Tarantula coefficients in localizing runtime anomalies in SOSs as demonstrated in [18], as well as [2, 44] for traditional software programs. This approach has also been adopted and studied by many researchers on fault localization for traditional software programs [6, 32, 37, 44].
- c_O-c_D *Ranking*. This approach ranks the BCs first by their Ochiai-based similarity coefficient scores then by their delay coefficient scores. The BCs are then inspected by decending order of their rankings.
- c_D-c_O *Ranking*. This approach is similar to the c_O-c_D approach, but it ranks BCs first by their delay coefficient scores then by their Ochiai-based similarity coefficient scores.
- $(c_D+c_O)/2$ *Ranking*. This approach ranks BCs by the average of their similarity coefficient scores and delay coefficient scores.

To measure the effectiveness of the anomaly localization approaches, we use the *localization cost*, i.e., the percentage of BCs of an SOS that must be inspected before localizing all faulty BCs. This measure has been widely used by other researchers [13, 44] for fault localization studies. *A low localization cost indicates high effectiveness and vice versa.* An approach with less localization cost can pinpoint the faulty BCs quicker and incurs less inspection expense, i.e., the costs for inspecting BCs during the localization process. For example, in Table 1, the ranking of N_6 (the faulty BC) is 3, the same as E_L and E_N. The system administrator can choose to inspect all the suspicious BCs at the same time or to inspect them in a random order. In a worst case scenario, the cost of pinpointing the anomaly is 3/22, or 13.64%. In the case of multiple anomalies, e.g., Table 2, our approach will inspect N_1, N_5, N_8, E_A, E_B, E_J, E_Q and E_R first, which are ranked 8 by their delay coefficients C_D. After that, N_6, E_L and E_N, which are ranked 11, are inspected. Thus, in the worst case scenario, the cost of pinpointing the faulty N_5 and N_6 with our approach is (8+3)/22=11/22, or 50.00%.

In order to evaluate our approach on different scales, we conduct 9 sets of experiments, where the component services of the SOS increases from 20 to 100 in steps of 10. Given a specific number of component services, the SOS is randomly structured with the sequence, branch and parallel structures discussed in Section 3.1. The loop structure is omitted because it will be transformed into branch structure anyway, as discussed in Section 3.1.

The response times of the BCs are generated according to different normal distributions based on a publicly available Web service dataset QWS [3, 4]. QWS comprises measurements of 9 QoS attributes (including response time) of over 2500 real-world Web services. The information about the services was collected from public UDDI registries, search engines and service portals. Their QoS values were measured using benchmark tools. At runtime, we generated and introduced a number of concurrent anomalies to randomly picked BCs to simulate a volatile operating environment. We increased the number of concurrent anomalies from 1 to 10 in steps of 1 in each set of experiments to simulate increasing levels of volatility in the operating environment. When anomalies occurred in BCs, delays that were randomly generated according to a normal distribution were applied to the corresponding BCs. In the real world, the delays caused by different anomalies to different BCs are usually different. There are short, medium and long ones, which delay the system to varying degrees. The long ones can significantly delay the system execution while the short ones can be neglected. There are many uncertain factors that may impact on the actual delay caused to a BC by an anomaly, e.g., the severity of the anomaly and the robustness of the BC. To simplify the application of delays and provide a generalised reference in the experiments, we generated delays caused by anomalies to different BCs according to a same normal distribution. This way, we consider or focus on only the "cost saving" effects from early anomaly localization enabled by our approach.

Upon runtime anomalies, we adopted *random inspection*, *c_O ranking approach* and our approaches to pinpoint the faulty BCs. All experiments were conducted on a machine with Intel i5-4570 CPU 3.20GHz and 8 GB RAM, running Windows 7 x64 Enterprise. For each set of experiment, we averaged the results obtained from 100 runs.

4.2 Experimental Results

4.2.1 Effectiveness

Table 4 presents the average localization costs of the 5 approaches across all 9 sets of experiments. On average, our three approaches, i.e., c_O-c_D, c_D-c_O and $(c_D+c_O)/2$, significantly outperform random inspection, by 13.1%, 18.2% and 18.1% respectively, compared to c_O ranking approach by 6.4%. In general, the advantages of our approaches over random inspection gradually decrease as the environmental volatility (indicated by the increasing number of concurrent anomalies) increases. However, their advantages over c_O ranking approach increase as the experimental volatility increases.

Fig. 6 compares the localization cost of our approaches (represented by *CO-CD*, *CD-CO* and (*CO*+*CD*)/2), with random inspection (represented by *RANDOM*) and the c_O ranking approach (represented by *CO*). In the case of a single anomaly, as expected, random inspection has to inspect approximately 50% of the BCs to pinpoint the faulty BC. In such scenarios, a single runtime anomaly does not lead to mutually-distinguisable end-to-end delays to different execution scenarios, making it impossible for our approaches to evaluate and utilise the severity of the end-to-end delays. As expected, c_O ranking approach and our approach demonstrate similar localization cost, between 12% and 27%, outperforming random inspection by significant margins.

As the number of concurrent anomalies increases, more BCs must be inspected to pinpoint all runtime anomalies. Accordingly, the localization costs of all approaches increase as the number of anomalies increases, rapidly at the beginning and gradually afterwards. As the number of concurrent anomalies increases, the probability that all the execution scenarios experience delays increases. Delays occurring in all execution scenarios will lead to all-1 delay vectors, making it hard for c_O ranking approach to pinpoint all the faulty BCs. Using not only the similarity coefficient but also the delay coefficient, our approaches can handle concurrent anomalies more efficiently, but still requires over 70% of the BCs to be inspected when there are 7 or more concurrent anomalies.

TABLE 4
AVERAGE LOCALIZATION COSTS
(ACROSS ALL SETS OF EXPERIMENTS)

# of Anomalies	Random	c_O	c_O-c_D	c_D-c_O	$\frac{c_O + c_D}{2}$
1	0.499	0.167	0.163	0.161	0.165
2	0.672	0.525	0.491	0.492	0.490
3	0.748	0.687	0.632	0.571	0.571
4	0.806	0.771	0.699	0.626	0.627
5	0.838	0.815	0.738	0.663	0.664
6	0.864	0.847	0.759	0.688	0.691
7	0.879	0.869	0.786	0.720	0.721
8	0.893	0.886	0.801	0.740	0.742
9	0.903	0.897	0.814	0.759	0.760
10	0.913	0.908	0.826	0.775	0.776
Average	0.802	0.737	0.671	0.619	0.621

An interesting obersevation is that, as the number of concurrent anomalies increases, the localization cost of c_O ranking approach gradually approximates that of the random inspection. The localization cost of c_O ranking approach catches up with that of the random inspection at certain points in all 9 sets of experiments, e.g., 6 in the first set of experiments, 7 in the fifth set and 8 in the ninth set. More concurrent anomalies indicate a more volatile operating environment and consequently a less reliable SOS. Thus, this oberservation shows that c_O ranking approach loses its advantage over random inspection in very volatile operating environments. However, our approaches still demonstrate significantly better effectiveness than random inspection. Compared with c_O ranking, our approaches achieve similar localization costs in relatively stable operating environments where only one or two concurrent anomalies are occurring in the SOS. However, their advantages over c_O ranking start to manifest as the number of concurrent anomalies increases. This is clearly demonstrated by the large gap below the red line in the cases of 2+ anomalies. Compared with c_O ranking, our approaches require much less BCs to be inspected to pinpoint all faulty BCs in volatile environments. In large-scale scenarios, this advantage can lead to significant saving of system administrative cost. For example, for an SOS that consists of 100 services, our c_D-c_O approach requires an average of 3 to 13 less services to be inspected than c_O ranking to localize all the anomalies. Our approaches can significantly speed up the anomaly detection process. Then, adaptation actions can be taken immediately to fix the system, relieving or eliminating the end-to-end system delay.

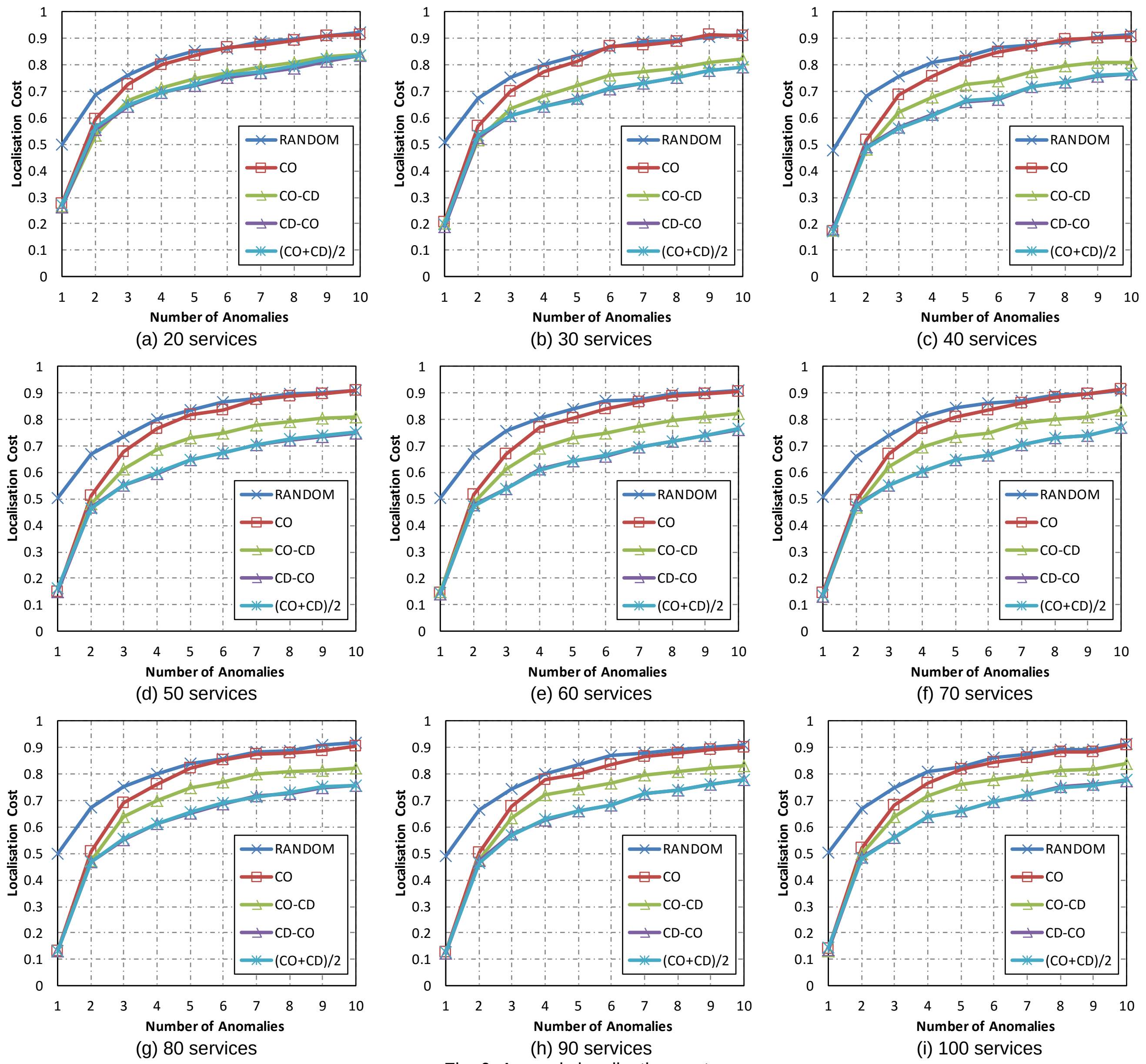


Fig. 6. Anomaly localization cost.

Although not explicitly demonstrated in Figure 5, it can be inferred that the localization costs of all approaches will reach 1.0 in an extreme environment where all BCs are faulty. In such a scenario, it does not make a difference which BCs to inspect first because all BCs are faulty.

Now we compare our own three approaches that rank BCs by their similarity coefficient scores and delay coefficient scores in different ways. The c_D-c_O and $(c_O+c_D)/2$ approaches demonstrate very similar performance, both outperforming the c_O-c_D approach. The advantages of the c_D-c_O and $(c_O+c_D)/2$ approaches over the c_O-c_D approach increase as the number of concurrent anomalies increase and as the size of the system increases. According to Table 4, the final winner is the c_D-c_O approach, with only marginal advantange over the $(c_O+c_D)/2$ approach.

To conclude, in a very stable operating environment, it does not make a big difference which anomaly localization approach is adopted. In a large-scale and volatile environment, our c_D-c_O approach has the best performance and is the most preferable choice over other approaches.

4.2.2 Efficiency

As discussed in Section 1, the response time plays an important role in the success of an SOS. A runtime localiastion approach must not take too much time to localize runtime anormalies occurring in an SOS because quick anomaly localization and rapid system adaptation are required to minimise the system delays perceived by the users of the system. An approach that can accurately rank the BCs according to their suspiciousness of being faulty is impractical if it needs to take a very long time. Here, we evaluate the efficiency of our approach, measured by the computation time taken to calculate the coefficients and rank the BCs, i.e., phases 4 and 5 in the anomaly localization procedure as presented in Fig. 3. The reason is that phases 1, 2 and 3 are performed offline at build-time while phases 4, 5 and 6 are performed at runtime and directly contribute to the total time consumption for anomaly localization upon runtime anomalies.

Our experimental results show that even in the largest-scale scenarios with SOSs sized 100 services, the coefficient calculation and BC ranking take only a few milliseconds to complete. This is expected because the spectrum-based anomaly localization, which is a statistical technique, has long been acknowledged as an extremely lightweight approach [44]. Detailed experimental results are not presented in this subsection as it does not provide any new and insightful findings about spectrum-based anormaly localization. Compared to c_O ranking, our approach takes only less than 1 millisecond extra computation time. Such a short extra computation time is negligible in most, if not all, real-world scenarios.

Based on the above discussion, we conclude that our approach is highly efficient and can be used to localize runtime anomalies in all types of SOSs, including time-constrained SOSs, e.g., interactive and multimedia SOSs. We also analyse the threat to validity of our evaluation, as presented in Appendix A.

5 Related Work

When operating in volatile environments, SOSs must be monitored in order to achieve timely and accurate detection and prediction of runtime anomalies. A great deal of monitoring techniques and approaches has been proposed for SOSs. Several languages are defined specifically for monitoring SOSs, such as WSCoL [7] and SECMOL [8]. These languages can be used to specify monitoring rules, constraints and assertions.

Many frameworks have also been proposed for monitoring SOSs. To name a few, Baresi et al. [9] propose a general and comprehensive solution for monitoring service compositions. In this monitoring solution, monitoring constraints can be defined on single and multiple instances, on punctual properties and on complete behaviours. Guinea et al. [15] propose a framework that integrates monitoring across the software and infrastructure layers. A variation of the MAPE control loop is introduced into the framework that acknowledges the multi-faceted nature of SOSs. Kertész et al. [27] integrate SALMon [36] in IMA4SSP - their monitoring approach to seamless service provisioning - so that it can collect dynamic reliability information of SOSs expressed in predefined quality metrics.

Monitoring only the critical parts of an SOS is a cost-effective solution [17] to SOS monitoring. As a result, non-critical BCs will not be monitored. When an anomaly occurs in such a BC, it must be localized quickly so that adaptation actions can be taken to fix the anomaly immediately in order to avoid or reduce the delay caused to the system execution [7, 10, 19].

Many fault localization techniques have been proposed for different types of software systems. To name a few, Artzi et al. [6] present how fault localization algorithms can be enhanced to pinpoint faults effectively in Web applications. Park et al. [37] present a dynamic fault localization technique that can pinpoint faults in concurrent programs. Novotný et al. [39] present a method for localizing faults occurring in service-oriented systems on mobile ad hoc networks. Sharma et al. [45] present a time-invariant relationships based approach for fault localization in distributed systems. Nguyen et al. [33] present FChain, a black-box online fault localization approach for diagnosing performance anomalies in cloud systems. It assumes that all the components of a cloud system are deployed within one cloud and thus all system metrics can be discovered immediately upon a detected performance anomaly. Thus, FChain is not suitable for SOSs whose component services are very often distributed across multiple clouds.

Compared to other fault localization techniques, including program slicing [49], delta debugging [31] and model-based diagnosis [29], spectrum-based fault localization is specifically designed to statistically calculate the likeliness of each software component being faulty based on information from passed and failed system runs [1]. It has been intensively studied in the context of traditional programming languages, such as C and Java, under the assumption that a component with a high similarity to the error vector has a higher probability of being the cause of the observed failure than a component with low similarity. There are many similarity coefficients that can be calculated to evaluate the suspiciousness of a software component being faulty [2]. The popular ones include the Jaccard coefficient, used by the Pinpoint tool [12], the coefficient used in the Tarantula fault localization tool [25] and the Ochiai coefficient c_O often used in the molecular biology domain [14]. Empirical performance comparison has been conducted among different similarity coefficients and the results showed that the Ochiai coefficient outperformed the Jaccard coefficient and the Tarantula coefficient [2, 44]. This has also been theorectically analysed and experimentally demonstrated by our previous work presented in [18] and [47]. Due to its high performance, Ochiai has been used in several faulty localization tools, e.g., Zoltar [22], GZoltar [11] and F^3 [23].

An SOS is different from a traditional standalone software system as it often operates in a distributed and volatile environment. Running test cases at build-time to localize the bugs in software systems, which is the method adopted by existing fault locsaliation techniques in the context of traditional programming languages, is impractical in localizaing runtime anomalies occurring in SOSs. An SOS usually consists of multiple distributed component services, whose states are difficult to inspect upon the occurrence of runtime anomalies. However, the execution engine of the SOS, e.g., the BPEL engine, is centralised. The end-to-end response time of the SOS can be inspected easily and end-to-end system delays can be detected efficiently. Taking this advantage, this paper presents a fault localization approach to attack the challenging research problem of runtime anomaly localization for SOSs. Our approach evaluates BCs' suspiciousness of being faulty by calculating their similarity coefficient scores and delay coefficient scores. The BCs that are more likely to be faulty are prioritised in the inspection.

6 Conclusions

In this paper, we propose a spectrum-based approach for anomaly localization in service-oriented systems. The main idea is to exploit end-to-end system quality data for localizing runtime anomalies. The system spectra are analysed to calculate the basic components' suspiciousness of resulting in the runtime anomaly. BCs' similarity coefficient scores and delay coefficient scores are calculated, which are used to rank the BCs. The comprehensive experimental analysis shows the effectiveness and efficiency of our approach, and proves that our approach can significantly save the time and efforts for runtime anomaly localization.

As part of future work, we plan to experiment with additional similarity metrics that can be used to rank BCs, and evaluate how their effectiveness compares to the approach presented in this paper. In addition, we will also investigate the statistical relation between localization cost and the number of execution scenarios.

Acknowledgment

This work is partly supported by Australian Research Council Discovery Project DP150101775 and the National Natural Science Foundation of China (Grant No. 61572375). Yun Yang is the corresponding author.

References

[1] R. Abreu, P. Druschel, and A. J. C. van Gemund, "On the Accuracy of Spectrum-based Fault Localization," *Proc of Testing: Academic and Industrial Conference - Practice And Research Techniques (TAICPART-Mutation 2007)*, Windsor, UK, pp. 89–98, 2007.

[2] R. Abreu, P. Zoeteweij, R. Golsteijn, and v. G. A. J.C., "A Practical Evaluation of Spectrum-Based Fault Localization," *The Journal of Systems and Software,* vol. 82, no. 11, pp. 1780–1792, 2009.

[3] E. Al-Masri and Q. H. Mahmoud, "Investigating Web Services on the World Wide Web," *Proc of 17th International Conference on World Wide Web (WWW 2008)*, Beijing, China, pp. 795-804, 2008.

[4] E. Al-Masri and Q. H. Mahmoud, "Qos-based Discovery and Ranking of Web Services," *Proc of 16th International Conference on Computer Communications and Networks (ICCCN07)*, pp. 529-534, 2007.

[5] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering (TSE),* vol. 33, no. 6, pp. 369-384, 2007.

[6] S. Artzi, J. Dolby, F. Tip, and M. Pistoia, "Fault Localization for Dynamic Web Applications," *IEEE Transactions on Software Engineering,* vol. 38, no. 2, pp. 314-335, 2012.

[7] L. Baresi and S. Guinea, "Self-Supervising BPEL Processes," *IEEE Transactions on Software Engineering,* vol. 37, no. 2, pp. 247-263, 2011.

[8] L. Baresi, S. Guinea, O. Nano, and G. Spanoudakis, "Comprehensive Monitoring of BPEL Processes," *IEEE Internet Computing,* vol. 14, no. 3, pp. 50-57, 2010.

[9] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, "Dynamo + Astro: An Integrated Approach for BPEL Monitoring," *Proc of IEEE International Conference on Web Services (ICWS2009)*, Los Angeles, CA, USA, pp. 230-237, 2009.

[10] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS Management and Optimisation in Service-Based Systems," *IEEE Transactions on Software Engineering (TSE),* vol. 37, no. 3, pp. 387-409, 2010.

[11] J. Campos, A. Riboira, A. Perez, and R. Abreu, "Gzoltar: An Eclipse Plug-in for Testing and Debugging," *Proc of 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 378-381, 2012.

[12] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. A. Brewer, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," *Proc of 2002 International Conference on Dependable Systems and Networks (DSN2002)*, Bethesda, MD, USA, pp. 595-604, 2002.

[13] H. Cleve and A. Zeller, "Locating Causes of Program Failures," *Proc of 27th International Conference on Software Engineering (ICSE 2005)*, St. Louis, Missouri, USA, pp. 342-351, 2005.

[14] A. da Silva Meyer, A. A. F. Garcia, A. P. de Souza, and C. L. de Souza, "Comparison of Similarity Coefficients Used for Cluster Analysis with Dominant Markers in Maize (Zea Mays L)," *Genetics and Molecular Biology,* vol. 27, no. 1, pp. 83-91, 2004.

[15] S. Guinea, G. Kecskemeti, A. Marconi, and B. Wetzstein, "Multi-layered Monitoring and Adaptation," *Proc of 9th International Conference on Service-Oriented Computing (ICSOC2011)*, Paphos, Cyprus, pp. 359-373, 2011.

[16] Q. He, J. Han, Y. Yang, H. Jin, J.-G. Schneider, and S. Versteeg, "Formulating Cost-Effective Monitoring Strategies for Services-based Systems," *IEEE Transactions on Software Engineering,* vol. 40, no. 5, pp. 461-482, 2014.

[17] Q. He, J. Han, Y. Yang, J.-G. Schneider, H. Jin, and S. Versteeg, "Probabilistic Critical Path Identification for Cost-Effective Monitoring of Cloud-Based Software Applications," *Proc of 9th International Conference on Service Computing (SCC2012)*, Honolulu, Hawaii, USA, pp. 178-185, 2012.

[18] Q. He, X. Xie, F. Chen, Y. Wang, R. Vasa, Y. Yang, and H. Jin, "Spectrum-Based Runtime Anomaly Localization in Service-Based Systems," *Proc of 12th IEEE International Conference on Services Computing (SCC 2015)*, New York City, NY, USA, pp. 90-97, 2015.

[19] Q. He, J. Yan, H. Jin, and Y. Yang, "Adaptation of Web Service Composition Based on Workflow Patterns," *Proc of 6th International Conference on Service-Oriented Computing (ICSOC2008)*, Sydney, Australia, pp. 22-37, 2008.

[20] Q. He, J. Yan, H. Jin, and Y. Yang, "Quality-Aware Service Selection for Service-based Systems Based on Iterative Multi-Attribute Combinatorial Auction," *IEEE Transactions on Software Engineering (TSE),* vol. 40, no. 2, pp. 192-215, 2014.

[21] G. Heward, J. Han, I. Müller, J.-G. Schneider, and S. Versteeg, "Optimizing the Configuration of Web Service Monitors," *Proc of 8th International Conference on Service-Oriented Computing (ICSOC2010)*, San Francisco, CA, USA, pp. 587-595, 2010.

[22] T. Janssen, R. Abreu, and A. J. van Gemund, "Zoltar: A Toolset for Automatic Fault Localization," *Proc of 2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 662-664, 2009.

[23] W. Jin and A. Orso, "Automated Support for Reproducing and Debugging Field Failures," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 24, no. 4, p. 26, 2015.

[24] J. A. Jones, J. F. Bowring, and M. J. Harrold, "Debugging in

Parallel," *Proc of International Symposium on Software Testing and Analysis (ISSTA 2007)*, London, United Kingdom, pp. 16-26, 2007.

[25] J. A. Jones and M. J. Harrold, "Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique," *Proc of 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, Long Beach, CA, USA, pp. 273-282, 2005.

[26] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A Self-adaptive Hierarchical Monitoring Mechanism for Clouds," *The Journal of Systems and Software,* vol. 85, no. 5, pp. 1029-1041, 2012.

[27] A. Kertész, G. Kecskeméti, A. Marosi, M. Oriol, X. Franch, and J. Marco, "Integrated Monitoring Approach for Seamless Service Provisioning in Federated Clouds," *Proc of 20th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP2012)*, Garching, Germany, pp. 567-574, 2012.

[28] R. Kohavi and R. Longbotham, "Online Experiments: Lessons Learned," *IEEE Computer,* vol. 40, no. 9, pp. 103-105, 2007.

[29] W. Mayer and M. Stumptner, "Model-Based Debugging–State of the Art and Future Challenges," *Electronic Notes in Theoretical Computer Science,* vol. 174, no. 4, pp. 61-82, 2007.

[30] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. C. Rinard, "Quality of Service Profiling," *Proc of 32nd ACM/IEEE International Conference on Software Engineering (ICSE2010)*, Cape Town, South Africa, pp. 25-34, 2010.

[31] G. Misherghi and Z. Su, "HDD: Hierarchical Delta Debugging," *Proc of 28th International Conference on Software Engineering*, pp. 142-151, 2006.

[32] L. Naish, H. J. Lee, and K. Ramamohanarao, "A Model for Spectra-Based Software Diagnosis," *ACM Transactions on Software Engineering and Methodology* vol. 20, no. 3, pp. 11:1 - 11:32, 2011.

[33] H. Nguyen, Z. Shen, Y. Tan, and X. Gu, "FChain: Toward Black-Box Online Fault Localization for Cloud Systems," *Proc of 33rd International Conference on Distributed Computing Systems (ICSCS 2013)*, Philadelphia, Pennsylvania, USA, pp. 21-30, 2013.

[34] OASIS, "*Web Services Business Process Execution Language Version 2.0,"* http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf. 2007.

[35] Object Management Group, "*Business Process Model And Notation (BPMN) Version 2.0,"* http://www.omg.org/spec/BPMN/2.0/PDF/. 2011.

[36] M. Oriol, J. Marco, X. Franch, and D. Ameller, "Monitoring Adaptable SOA-Systems using SALMon," *Proc of Workshop on Service Monitoring, Adaptation and Beyond (Mona+), ServiceWave 2008*, Madrid, Spain, pp. 19-28, 2008.

[37] S. Park, R. W. Vuduc, and M. J. Harrold, "Falcon: Fault Localization in Concurrent Programs," *Proc of 32nd ACM/IEEE International Conference on Software Engineering*, Cape Town, South Africa, pp. 245-254, 2010.

[38] M. Pathirage, S. Perera, I. Kumara, D. Weerasiri, and S. Weerawarana, "A Scalable Multi-Tenant Architecture for Business Process Executions," *International Journal of Web Services Research,* vol. 9, no. 2, pp. 21-41, 2012.

[39] A. L. W. Petr Novotný, Bong-Jun Ko:, "Fault Localization in MANET-Hosted Service-Based Systems," *Proc of IEEE 31st Symposium on Reliable Distributed Systems (SRDS 2012)*, Irvine, CA, USA, pp. 243-248, 2012.

[40] E. Piel, A. Gonzalez-Sanchez, H.-G. Gross, A. J. C. van Gemund, and R. Abreu, "Online Spectrum-based Fault Localization for Health Monitoring and Fault Recovery of Self-Adaptive Systems," *Proc of 8th International Conference on Autonomic and Autonomous Systems (ICAS2012)*, Brisbane, Australia, pp. 64-73, 2012.

[41] É. Piel, A. Gonzalez-Sanchez, H.-G. Gross, A. J. C. van Gemund, and R. Abreu, "Online Spectrum-based Fault Localization for Health Monitoring and Fault Recovery of Self-Adaptive Systems," *Proc of 8th International Conference on Autonomic and Autonomous Systems (ICAS 2012)*, St. Maarten, The Netherlands Antilles, pp. 25-30, 2012.

[42] T. W. Reps, T. Ball, M. Das, and J. R. Larus, "The Use of Program Profiling for Software Maintenance with Applications to the Year 2000 Problem," *Proc of 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/SIGSOFT FSE)*, Zurich, Switzerland, pp. 432-449, 1997.

[43] R. M. H. Ron Kohavi, Dan Sommerfield, "Practical Guide to Controlled Experiments on the Web: Listen to Your Customers Not to the Hippo," *Proc of 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Jose, California, USA, pp. 959-967, 2007.

[44] R. A. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold, "Lightweight Fault-Localization Using Multiple Coverage Types," *Proc of 31st International Conference on Software Engineering (ICSE 2009)*, Vancouver, Canada, pp. 56-66, 2009.

[45] A. B. Sharma, H. Chen, M. Ding, K. Yoshihira, and G. Jiang, "Fault detection and localization in distributed systems using invariant relationships," *Proc of 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013)*, Budapest, Hungary, pp. 1-8, 2013.

[46] C. Verbowski, E. Kiciman, A. Kumar, B. Daniels, S. Lu, J. Lee, Y.-M. Wang, and R. Roussev, "Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management," *Proc of 7th Symposium on Operating Systems Design and Implementation (OSDI2006)*, Seattle, WA, USA, pp. 117-130, 2006.

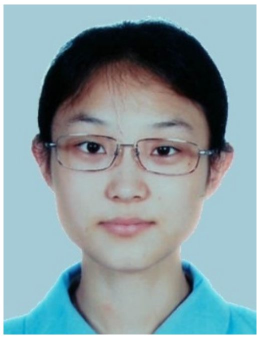
[47] X. Xie, T. Y. Chen, F.-C. Kuo, and B. Xu, "A Theoretical Analysis of the Risk Evaluation Formulas for Spectrum-Based Fault Localization," *ACM Transactions on Software Engineering and Methodology,* vol. 22, no. 4, pp. 31:1-31:40, 2013.

[48] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering (TSE),* vol. 30, no. 5, pp. 311-327, 2004.

[49] X. Zhang, S. Tallam, N. Gupta, and R. Gupta, "Towards Locating Execution Omission Errors," *ACM Sigplan Notices,* vol. 42, no. 6, pp. 415-424, 2007.

[50] Z. Zheng and M. R. Lyu, "Collaborative Reliability Prediction of Service-Oriented Systems," *Proc of 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, Cape Town, South Africa, pp. 35-44, 2010.

Qiang He received the his first Ph. D. degree from Swinburne University of Technology (SUT), Australia, in 2009 and his second Ph. D. degree in computer science and engineering from Huazhong University of Science and Technology (HUST), China, in 2010. He is a lecturer at Swinburne University of Technology. His research interests include software engineering, cloud computing and services computing.



Xiaoyuan Xie received her PhD degree from Swinburne University of Technology, Australia in 2012. She is a professor in Computer School, Wuhan University, China. Her research interests include software analysis, testing, debugging, search-based software engineering and cloud computing.

Yanchun Wang received his M.Eng. degree in computer science from Beihang University, Beijing, China, in 2006. He is currently a Ph.D. student at Swinburne University of Technology, Australia. His research interests include services computing and cloud computing.

Dayong Ye received his MSc and PhD degrees both from University of Wollongong, Australia, in 2009 and 2013, respectively. He is a research fellow at Swinburne University of Technology, Australia. His research interests focus on service-oriented computing, self-organisation and multi-agent systems.

Feifei Chen received her PhD degree from Swinburne University of Technology, Australia in 2015. Her research interests include software engineering, cloud computting and green computing.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. Jin received his PhD in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is a Fellow of CCF, senior member of the IEEE and a member of the ACM. He has co-authored 22 books and published over 700 research papers. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Yun Yang received the PhD degree from the University of Queensland, Australia in 1992. He is a full professor at Swinburne University of Technology. His research interests include software engineering, cloud computing, workflow systems and service-oriented computing.