He, Q., Han, J., Yang, Y. et al. (2012). Probabilistic critical path identification for cost-effective monitoring of service-based systems.


Available from: http://dx.doi.org/10.1109/SCC.2012.39.

Copyright © 2012 IEEE.

This is the author’s version of the work, posted here with the permission of the publisher for your personal use. No further distribution is permitted. You may also be able to access the published version from your library. The definitive version is available at http://ieeexplore.ieee.org/.
Probabilistic Critical Path Identification for Cost-Effective Monitoring of Service-based Systems

Qiang He, Jun Han, Yun Yang and Jean-Guy Schneider
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Australia 3122
{qhe, jhan, yyang, jschneider}@swin.edu.au

Hai Jin
Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, China 430074
hjin@hust.edu.cn

Steve Versteeg
CA Labs
Melbourne, Australia
steve.versteeg@ca.com

Abstract—When operating in volatile environments, service-based systems (SBSs) that are built through dynamic composition of component services must be monitored in order to guarantee the response times of the SBSs. In particular, the critical path of a composite SBS, i.e., the execution path in the service composition with the maximum execution time, should be prioritised in cost-effective monitoring as it determines the response time of the SBS. In volatile operating environments, the critical path of a SBS is probabilistic. As such, it is important to estimate the criticalities of the execution paths and the component services, i.e., the probabilities that they are critical, to decide which parts of the system to monitor. In this paper, we propose a novel approach to the identification of Probabilistic Critical Path for Service-based Systems (PCP-SBS). PCP-SBS takes into account the probabilistic nature of the critical path and calculates path criticalities in the context of service composition. We evaluate PCP-SBS experimentally using SBSs that are synthetically composed based on a real-world Web service dataset.

Keywords—Service composition; Web service; monitoring; critical path

I. INTRODUCTION

Cloud Computing provides on-demand provisioning of software applications [1]. With the proliferation of the Software-as-a-Service (SaaS) delivery model and the pay-as-you-go business model promoted by Cloud Computing, it is expected that more and more businesses will develop their software applications by composing in-house services (their core business functions) and external services (non-core business functions) [2], which are locally or remotely accessed by application engines (e.g., BPEL engines [12]).

Built through dynamic compositions of loosely coupled component services offered by independent (and often distributed) providers, SBSs are operating in environments where key characteristics of the component services, such as the quality-of-service (QoS) properties, tend to be volatile. During the execution of a SBS, various runtime anomalies may occur that impact the quality of the SBS, e.g., unexpected workload changes, errors in the component services and failures of data transmission. In this context, how to manage the quality of a SBS by detecting and adapting to runtime anomalies has become a promising research direction [3-5].

The response time, among various QoS properties, is of particular significance and challenge in QoS management in SBSs. 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 their other QoS dimensions [6]. A video encoding application, for example, can often produce higher quality video if it is given more time to encode frames. In addition, the management of other QoS dimensions is tightly coupled with response time management. During the execution of a SBS, it often needs to be adapted to accommodate runtime anomalies that may jeopardise its quality. The adaptation itself consumes time, and as a result, contributes to delaying the execution of the SBS. In particular, runtime anomalies that occur on the critical path of the service composition, i.e., the execution path with the maximum execution time, will cause delays that directly impact the response time of the SBS. Thus, timely detection and prediction of runtime anomalies, especially those that occur on the critical path, are significant to effective response time management in SBSs.

Monitoring, as an essential part of Service-Oriented Architecture and SBSs [3, 4, 7], is required for QoS management in SBSs. By monitoring the execution of the basic components (BCs) that compose a SBS, i.e., the component services and the data transmissions between the component services, runtime anomalies can be predicted or detected. Then, adaptation actions can be taken to fix those anomalies and update the SBS to better manage and guarantee its quality. A straightforward solution to timely detection and prediction of runtime anomalies is to constantly monitor all the BCs of the SBS. However, monitoring consumes resources, including software, hardware and sometimes human resources. Thus, constantly monitoring all the BCs has a number of shortcomings, including potentially excessive monitoring resource consumption and poor scalability, especially for SBSs that...
provide continuous services, e.g., multimedia streaming [8]. Particularly, in the cloud environments, a cloud service provider may maintain up to hundreds of thousands of services for their clients [2]. In such a context, the issue of monitoring cost, i.e., the cost of monitoring in terms of monitoring resources, is particularly critical. For those cloud service providers, constantly monitoring all the services is very expensive. It is to their best benefits to maintain the monitoring cost at a reasonable and affordable level while being able to guarantee the response times of their applications. Therefore, cost-effective monitoring for SBSs has become a significant issue, which has two aspects: 1) given limited monitoring resources, how to make the best use of it to monitor a SBS; 2) given different levels of requirements for the response time of a SBS, how to allocate minimum monitoring resources to meet the requirements.

The key to cost-effective monitoring of a SBS is the identification and monitoring of the critical path of the service composition because any delays on the critical path will directly impact the response time of the SBS. However, the volatility of the operating environments makes the response times of the BCs of the SBSs probabilistic [4]. Thus, the critical path of a SBS is probabilistic - every execution path can be critical with certain probabilities which represent their criticalities in the service composition. Thus, identifying the critical path turns into calculating those probabilities. With the knowledge of the criticalities, cost-effective monitoring can be conducted by prioritising the execution paths with high criticalities in monitor allocation.

In this paper, we present a novel approach to the identification of Probabilistic Critical Path for SBSs (PCP-SBS). It includes: 1) a timing model that captures the probabilistic nature of the timing properties of the BCs; and 2) a method for calculating the criticalities of the execution paths and the BCs. Experimental results demonstrate that the response times of the SBSs can be managed in a cost-effective manner by conducting PCP-SBS based monitoring. The major contributions of this paper include:

- We capture the probabilistic nature of the SBSs in volatile operating environments. The proposed model and method allow developers to evaluate and analyse SBSs in a more realistic way.

- We propose a method for the identification of probabilistic critical path, which helps developers calculate the criticalities of different execution paths and BCs of SBSs. Based on the criticality information, cost-effective monitoring can be conducted for the response time management of SBSs.

- To evaluate PCP-SBS, we conduct extensive experiments using SBSs synthetically composed based on a published dataset which contains QoS information about over 2500 real-world Web services. The experimental results demonstrate that PCP-SBS based monitoring consumes significantly less (55.67% on average) monitoring resources than random monitoring.

The rest of the paper is structured as follows. The next section motivates this research with an example. Section III introduces the composition model for SBSs. Section IV presents the proposed approach to the identification of probabilistic critical path for SBSs. Experimental evaluation is given in Section V. Section VI reviews related work. Finally, Section VII summarises the major contributions of this paper and outlines future work.

II. MOTIVATING EXAMPLE

This section presents an example SBS, namely OnlineLive to motivate this research. As presented in Figure 1, this application consists of 24 BCs and offers a live-on-demand service to convert, subtitle and transmit various live video streams for clients.

$N_0$ and $N_6$ are virtual nodes that represent the entry and exit points of OnlineLive. $N_1, N_2, ..., N_8$ represent the component services and $E_0, E_1, ..., E_8$ represent the data transmissions between the component services. In response to a client’s request, the execution process of OnlineLive is described as follows: Step 1) $N_1$ splits the live media stream selected by the client into separate video and audio streams. Step 2) $N_3$ and $N_4$ convert the video stream and audio stream into the formats that are compatible with the client’s end device. Meanwhile, $N_5$ generates the subtitle by performing speech recognition on the audio stream. Then, based on the client’s preference or current 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) $N_7$ produces a media stream by merging the converted video stream, audio stream and the translated subtitle. Step 4) The media stream is transmitted to the client.

The live media stream is continuously processed and transmitted to the client as packets so that the client can receive a continuous media stream. Operating in a volatile environment, OnlineLive has the following characteristics:

- **OnlineLive** must timely and continuously process the media stream. Otherwise, the client will receive a jittering media stream. Thus, **OnlineLive** must be monitored so that runtime anomalies that may cause delays to **OnlineLive** can be detected or predicted on time. Then, adaptation actions can be taken immediately to reduce or prevent the delays by fixing the anomalies.

- **OnlineLive** has four end-to-end execution paths: $EP_1$, $EP_2$, $EP_3$ and $EP_4$. The one with the maximum execution time determines the response time of **OnlineLive**. However, various anomalies may occur at runtime, causing delays to different execution paths. As a result, the critical path of **OnlineLive** may change at runtime.
Statistically, each of the four execution paths can be critical with certain probabilities.

- The critical path of OnlineLive must be monitored with the highest priority. However, adjusting the monitors dynamically at runtime as the critical path changes is impractical, especially in highly volatile environments. It is often difficult for the adjustment of the monitors to keep up with the pace of the critical path change. In addition, frequent adjustment of monitors can be very expensive in terms of software, hardware and sometimes human resources.

Given the above characteristics, in order to perform cost-effective monitoring for response time management of OnlineLive, monitors need to be properly allocated based on the criticalities of the execution paths and BCs. BCs on execution paths with higher criticalities should be monitored with higher priorities.

III. COMPOSITION MODEL

This section introduces the composition model adopted in this research for representing and analysing SBSs.

A. Compositional Structures

Compositional structures describe the order in which the component services are executed in a service composition to realise the functionality of a SBS. There are four types of basic compositional structures, i.e., sequence, branch, loop and parallel [9, 10], which are included in BPMN [11] and addressed by BPEL [12] (the de facto standards for specifying service-oriented business processes).

- **Sequence.** In a sequence structure, the BCs are executed one by one.
- **Parallel.** In a parallel structure, all the branches are executed at the same time.
- **Branch.** In a branch structure, only one branch is selected for execution. For every set of branches \(\{b_1, \ldots, b_n\}\), the execution probability distribution \(p(b_1), \ldots, p(b_n)\) is specified, where \(p(b_i), i=1, \ldots, n\), is the probability that the \(i^{th}\) branch is selected for execution.
- **Loop.** In a loop structure, the loop is executed for \(n (n \geq 0)\) times. For every loop, the probability distribution \(p_{n0}, \ldots, p_{nMNI}\), \((0 \leq p_{ni} \leq 1, \sum_{i=0}^{MNI} p_{ni}=1.0)\) is specified, where \(p_{ni}, i=0, \ldots, MNI\), is the probability that the loop iterates for \(i\) times and \(MNI\) is the expected maximum number of iterations for the loop.

\(p(b_i), p_{ni}\) and \(MNI\) can be evaluated based on the past executions of the SBS or can be specified by the developer [10, 13]. We assume that for every loop, the \(MNI\) is determined or estimated. Otherwise, if an upper bound for the number of iterations for a loop does not exist, the execution times of the execution paths that contain the loop cannot be calculated since the loop may iterate infinitely.

In this research, we represent service compositions by UML activity diagrams, where activity nodes represent component services and edges represent data transmissions. We assume that a service composition is characterised by only one entry point and one exit point, and it only includes structured loops with only one entry point and one exit point. If a service composition includes loop structures, we transform them into branch structures by representing loop iterations as a sequence of branches with corresponding execution probabilities [10].

B. Execution Scenarios

In a service composition that involves branches or loops, multiple possible execution scenarios can be identified. These execution scenarios do not contain branch or loop structures, and hence can be modelled as Directed Acyclic Graphs (DAGs). As an example, Figure 2 present the two possible execution scenarios identified from OnlineLive: \(es_1=\{EP_1, EP_2, EP_3\}\) and \(es_2=\{EP_1, EP_2, EP_4\}\). The criticality evaluation of the execution paths must consider all the possible execution scenarios according to their execution probabilities, i.e., the appearance probabilities of the execution scenarios. Thus, we need to calculate the execution probability of each possible execution scenario identified from the service composition. The execution probability of an execution scenario is the product of the execution probabilities of all branches in the execution scenario that belong to branch structures in the service composition. For example, in Figure 2, there is \(ep(es_1)=p(b_1)\) and \(ep(es_2)=p(b_2)\).

IV. CRITICALITY EVALUATION

In this section, we present the proposed approach to criticality evaluation for SBSs, including a probabilistic timing model and methods for dominance probability calculation and criticality calculation.

A. Timing Model

In volatile environments, the execution of the BCs of a SBS often suffers impacts of various runtime anomalies. Thus, the timing properties of the BCs are random variables instead of deterministic values, which makes the response time of the SBS probabilistic [4]. For realistic evaluation of the response time of a SBS, a model is needed that takes into account the probabilistic nature of the timing properties of the BCs that compose the SBS. In this research, three types of timing properties are considered:
• Start time \((T_0)\): the time elapsed since the moment when the SBS is invoked (referred to as \textit{time zero}) until the moment when the BC is activated.

• Response time \((T_R)\): the time elapsed for the BC to complete since its start time.

• Finish time \((T_F)\): the time elapsed for the BC to complete since time zero.

The response time of a BC \(S_i\) is expressed in the standard form as:

\[
T_{SP}(S_i) = t_0 + \sum_{i=1}^{n} w_i \cdot \Delta X_i
\]

where \(t_0\) is the mean value of \(T_{SP}(S_i)\), \(\Delta X_i\), \(i=1, \ldots, n\), represent the variation of \(n\) sources of anomaly \(X_i\), \(i=1, 2, \ldots, n\), from their mean values; \(w_i\), \(i=1, \ldots, n\), represent the sensitivities of \(S_i\) to each of the sources of anomaly.

For the response time of a BC \(S_i\), i.e., \(T_{SP}(S_i)\), \(t_0\), \(w_i\), and the distributions of \(\Delta X_i\) can be evaluated by inspecting \(S_i\)'s past executions, clients’ feedbacks, service providers' profiles, etc. Given a service composition \(SC\)=\{\(S_1\), \(S_2\), \ldots, \(S_n\)\} and \(T_{SP}(S_i)\), \(i=1, \ldots, n\), the start times and finish times of the BCs, denoted by \(T_{SP}(S_i)\) and \(T_{TF}(S_i)\), \(i=1, \ldots, n\), where \(T_{SP}(S_i)\)=\(T_{SP}(S_i)\)+\(T_{TF}(S_i)\), can be calculated by a forward timing property propagation (see Section IV.B).

B. Dominance Probability Calculation

Given two random timing properties \(T_A\) and \(T_B\), the dominance probability of \(T_A\) over \(T_B\), noted as \(D_{T_A}(T_B)\), is the probability that \(T_A\) is larger than or equal to \(T_B\), where \(D_{T_A}(T_B)\in[0, 1]\) and \(D_{T_A}(T_B)=1-D_{T_B}(T_A)\). Given \(n\) timing properties \(T_i\), \(i=1, \ldots, n\), the dominance probability of each, noted as \(D(T_i)\), is the probability that it is larger than or equal to all the others. The calculation of \(D(T_i)\) depends on the distributions of \(\Delta X_i\). In this research, we assume that \(\Delta X_i\) is subject to Gaussian distributions to facilitate general evaluation of timing properties. For other probability distributions, e.g., exponential distribution, corresponding techniques can be adopted for the calculation of dominance probability [14].

Let us first consider the case of two independent timing properties \(T_A\) and \(T_B\):

\[
T_A = t_A + \sum_{i=1}^{n} a_i \cdot \Delta X_i
\]

\[
T_B = t_B + \sum_{i=1}^{n} b_i \cdot \Delta X_i
\]

Their expected values are: \(\mu_A=E[T_A]=t_A\) and \(\mu_B=E[T_B]=t_B\), and their variances are:

\[
\sigma_A^2 = \text{var}(T_A) = E[(T_A - E(T_A))^2] = \sum_{i=1}^{n} a_i^2
\]

\[
\sigma_B^2 = \text{var}(T_B) = E[(T_B - E(T_B))^2] = \sum_{i=1}^{n} b_i^2
\]

Since \(\Delta X_i\), \(i=1, 2, \ldots, n\), are subject to Gaussian distributions, there is \(T_A \sim N(\mu_A, \sigma_A^2)\), \(T_B \sim N(\mu_B, \sigma_B^2)\) and \(Y \sim \text{var}(Z)\), \(Z=(1-D_{T_B}(T_A))+(\mu_B-\mu_A)\), \(D_{T_A}(T_B)\) and \(D_{T_B}(T_A)\) can be calculated by:

\[
D_{T_A}(T_B) = P(T_A - T_B \geq 0) = P(Y \leq 0) = F_Y(0)
\]

\[
\mu_Y = t_B - t_A
\]

\[
\sigma_Y^2 = \sum_{i=1}^{n} a_i^2 + \sum_{i=1}^{n} b_i^2
\]

where \(F_Y\) is the cumulative probability function of \(Y\).

Now let us consider the case of multiple independent timing properties \(T_{n_1}, T_{n_2}, \ldots, T_{n_n}\). Let \(Z = \max(T_{n_1}, T_{n_2}, \ldots, T_{n_n})\), there is:

\[
D(T_i) = P(T_i \geq Z) = P(T_i - Z \leq 0) = F_{Z-T_i}(0)
\]

\[
F_{Z-T_i}(0)
\]

where \(Z\) is the cumulative probability function of \(Y\).

Before applying formula (6), we appeal to [15] for an analytic expression of \(Z\).

Definition 1. Dominance Probability of a BC: For a given BC \(S_i\) in an execution scenario \(ES_i\)=\{\(S_1\), \(S_2\), \ldots, \(S_n\)\} of a service composition, the dominance probability of \(S_i\), denoted as \(D(S_i)\), is the probability that \(S_i\)'s finish time solely determines the start time of its succeeding BC\(s\).

The dominance probability computation and timing property propagation for the BCs through an execution scenario of the service composition are interleaved based on their timing properties and the compositional structures that they are involved in. Since an execution scenario does not contain branch or loop structures, we only need to consider sequence and parallel structures.

• Sequence. A BC \(S_i\) has only one precedent BC \(S_j\) in a sequence structure. Thus, the start time of \(S_i\) is solely determined by \(S_j\)'s finish time:

\[
T_{DP}(S_i) = T_{DP}(S_j)
\]

According to Definition 1, there is \(D(S_j)=1.0\). In Figure 2(a), the start time of \(N_1\) is solely determined by the finish time of \(E_4\), i.e., \(T_{DP}(N_1)=T_{DP}(E_4)\). Thus, there is:

\[
D(E_4)=1.0
\]

• Parallel. In the splitting part of a parallel structure where a precedent edge \(E_p\) splits into multiple edges \(E_1, \ldots, E_n\), the start times of \(E_i\), \(i=1, \ldots, n\), are solely determined by the finish time of \(E_p\). Thus, formulas (14) and (15) can be applied to calculate \(T_{DP}(E_1)\) and \(D(E_1)\). In Figure 2(a), the start time of \(E_5\), \(E_6\) and \(E_7\) are solely determined by the finish time of \(E_5\), i.e., \(T_{DP}(E_5)\)=\(T_{DP}(E_5)\)=\(T_{DP}(E_6)\). Hence, there is \(D(E_5)=1.0\). In the
merging part of a parallel structure where multiple edges $E_1$, ..., $E_n$ merge into an edge $E$, that succeeds the branches, the start time of $E$ is determined by the merging edge that finishes the last, i.e., the merging edge that has the maximum finish time. For example, in Figure 2(a), there is $T_D(E_N) = \max(T_D(E_1), T_D(E_2), T_D(E_3))$. Since the finish times of the merging edges are random variables, the start time of $E_N$ is probabilistically determined by each merging edge’s finish time according to their respective dominance probabilities: $T_D(E_N) = \sum_{i=1}^{n} T_F(E_i) \times P_D(E_i)$ (16)

where $\sum_{i=1}^{n} D(E_i) = 1$ and $D(E_i)$, $i=1$, ..., $n$, can be obtained by calculating the dominance probabilities of $E_i$’s finish time using formulas (6) through (13).

To proceed with the interleaved timing property propagation and dominance probability computation, we use the weighted average value, denoted by $T_D(N)$, as the finish time of the succeeding edge $E$:

$$T_D(N) = \sum_{i=1}^{n} T_D(E_i) \times T_F(E_i)$$ (17)

For example, in Figure 2(a), there is $T_D(E_N) = D(E_0) \times T_F(E_1) + D(E_2) \times T_F(E_2) + D(E_3) \times T_F(E_3)$ and $T_D(E_4) = T_F(E_4) + D(E_0) \times T_F(E_1) + D(E_2) \times T_F(E_2) + D(E_3) \times T_F(E_3)$.

For other BCs in a parallel structure, formulas (14) and (15) apply as each parallel branch is essentially a sequence structure.

In particular, in a parallel structure, we consider the response time of the edges that precedes and succeeds the branches, e.g., $E_0$ and $E_N$ in Figure 2(a), as a constant in the timing property propagation. In Figure 2(a), given $T_D(E_0) = 0$, $T_F(E_1) = T_F(E_2) = T_F(E_3) = T_F(E_4) = T_F(N)$.

By interleaving the timing property propagation and the dominance probability computation, the dominance probabilities of the BCs in an execution scenario can be calculated. In Figure 3, we present the dominance probabilities (denoted as $D$) of the BCs in execution scenario 1 of OnlineLive, which are calculated based on arbitrarily chosen timing properties as a demonstrative example. In reality, the dominance probabilities of the BCs are calculated based on their real timing properties.

### C. Criticality Calculation

The static concept of critical path of a service composition is the execution path with the maximum execution time [13]. However, in volatile operating environments, the response times of the BCs of a SBS are seldom static. Instead, they are subject to change at runtime. Thus, every execution path in the service composition can be critical with certain probabilities. Hence, the problem of identifying the critical path of a service composition turns into the problem of calculating those criticalities. This section formally defines criticality and presents the methods for criticality calculation.

First of all, we give the formal definitions of path criticality, node criticality and edge criticality.

**Definition 2. Path Criticality:** For a given execution path $EP_i$ in a service composition $SC = \{EP_1, \, EP_2, \ldots, \, EP_n\}$, the criticality of $EP_i$, noted as $C(EP_i)$, is the probability that $EP_i$ is the critical path, i.e., $EP_i$ has the maximum execution time among $EP_1, \ldots, EP_n$.

**Definition 3. BC Criticality:** For a given BC $BC_i$ in a service composition $SC = \{S_1, \ldots, S_n\}$, the criticality of $BC_i$ noted as $C(S_i)$, is the probability that $S_i$ belongs to the critical path.

As introduced in Section III.B, multiple execution scenarios can be identified from a service composition. To evaluate the criticalities of the execution paths and the BCs in a service composition, first we need to calculate their criticalities in each execution scenario with a backward propagation through the execution scenario starting with assigning 1.0 to the criticality of the exit node, e.g., $N_i$ in Figure 3, as it always belongs to the critical path. Then, the criticality calculation can be performed following certain rules.

**Rule 1:** The criticality of an execution path in an execution scenario is the product of the dominance probabilities of all the edges that belong to the execution path.

For example, in Figure 3 for example, for $EP_1$, to be critical, $E_0$, $E_3$, $E_5$, $E_7$, and $E_0$ have to determine the start time of $N_1$, $N_2$, $N_3$, $N_4$, and $N_5$ respectively. Thus, the criticalities of $EP_1$ is calculated as: $C_{EP_1}(EP_1) = D(E_0) \times D(E_3) \times D(E_5) \times D(E_7) \times D(E_0) = 0.5$.

**Rule 2:** The criticality of a node in an execution scenario equals to the criticality of its succeeding edge.

As each node has only one succeeding edge in an execution scenario, the criticality of a node depends on how critical its succeeding edge is. In Figure 3, there is $C(N_1) = C(N_2) = 1.0$ and $C(N_3) = C(N_4) = 0.3$.

**Rule 3:** The criticality of an edge in an execution scenario is the product of its dominance probability and the sum of the criticalities of its succeeding BCs.

An edge may have one or many succeeding BCs. Hence, for an edge to be critical it has to determine the start times of its succeeding BC(s) and its succeeding BC(s) have to be critical. For example, in Figure 3, there is $C(E_0) = D(E_0) \times (C(E_3) + C(E_5) + C(E_7)) = 1.0 \times (0.5 + 0.3 + 0.2) = 1.0$ and $C(E_0) = D(E_0) \times (C(N_3) + C(N_4)) = 1.0 \times 0.2 = 0.2$.

Now, the criticality of an execution path $EP_i$ or a PC $S_i$ in the service composition can be computed by a weighted average over their criticalities obtained in all the execution scenarios using the execution scenarios’ execution probabilities as weights:

$$C(EP_i) = \sum_{j=1}^{n} \epsilon_{S_j} \times C_{S_j}(EP_i)$$ (18)
where $ep(e_j)$ is the execution probability of $e_j$ and $C_{es_j}(S_j)$ is the criticality of $S_j$ in $e_j$.

V. EXPERIMENTS

To evaluate PCP-SBS, we have developed a prototype tool that implements PCP-SBS in Java using JDK 1.6.0 and Eclipse Java EE IDE. Using this prototype tool, we have conducted a range of in-lab experiments in a simulated volatile environment, aiming at evaluating the effectiveness, cost-effectiveness and computational overhead of PCP-SBS.

A. Experimental Setup

In the experiments, the evaluation process mimicked the OnlineLive example presented in Section II, which consisted of 24 BCs. The response times of the BCs were generated according to different normal distributions based on a publicly available Web service dataset QWS [16]. QWS comprises measurements of nine QoS properties (including response time) of over 2500 real-world Web services.

During the execution of OnlineLive, a certain number of runtime anomalies were generated based on the fault rate and randomly introduced to the BCs to simulate volatile environments. In order to create different levels of volatility in the experimental environment, we increased the fault rate from 10% to 40% in steps of 10%. When anomalies occurred to unmonitored BCs, delays that were randomly generated based on normal distributions were applied to the BCs. If a BC was being monitored, the caused delay was avoided (representing the fact that the anomaly was detected or predicted and adaptation actions were taken on time to fix the anomaly).

Three sets of experiments were conducted in each volatile environment. In each set of experiments, 1,000 OnlineLive instances were run and the response times of OnlineLive were averaged. In set #1, no monitors were allocated. In set #2, the monitors were randomly allocated to the BCs - a naive approach for monitor allocation that does not consider the criticalities of the execution paths and the BCs. In set #3, the monitors were allocated according to the criticalities of the execution paths from high to low, first the execution path with the highest criticality, then the second highest, etc. When the remaining monitoring resources were not enough to cover an entire execution path, the BCs with the highest criticalities on that execution path were monitored first. Given the criticalities of the BCs as weights in a DAG, we adopt the method proposed in [17] for critical path enumeration, which runs in $O(m+n\log n+k)$ to find $k$ longest execution paths in a service composition that consists of $n$ nodes and $m$ edges. In sets #2 and #3, the monitoring coverage, i.e., the maximum proportions of BCs that were monitored, was increased from 0% to 100% in steps of 10% to simulate scenarios with different levels of available monitoring resources. In this way, we are able to evaluate the cost-effectiveness of PCP-SBS.

In the experiments for computational overhead evaluation, we simulated SBSs that comprised different numbers of BCs, from 24 to approximately 10,000. Then, we used the prototype tool to perform dominance probability and criticality evaluation. By comparing the computation time, we are able to evaluate the computational overhead of PCP-SBS. The experiments were conducted on a machine with AMD Athlon(tm) X4 640 3.00GHz CPU and 8 GB RAM, running Windows 7 x64 Ultimate.

B. Experimental Results

In this section, we present the evaluation results of the effectiveness, cost-effectiveness and computational overhead of the proposed approach.

Figure 4 demonstrates the average response time of OnlineLive obtained in different volatile environments. As the fault rate increases, the average response time of OnlineLive increases because more anomalies cause longer total delay to OnlineLive. Random monitoring and PCP-SBS based monitoring improved the response time of OnlineLive by an average of 17.87% and 27.80% respectively across all experimental cases. The results presented in Figure 4 also provide the knowledge as to how to cost-effectively allocate monitors to meet different levels of requirements for the response time of OnlineLive. Take Figure 4(d) for example, to make sure the average response time of OnlineLive is shorter than 30 seconds, random monitoring requires at least 80% monitoring coverage while PCP-SBS based monitoring only requires a minimum of 40% monitoring coverage. When the monitoring coverage is equivalent to and larger than 50%, the average response time of OnlineLive obtained by PCP-SBS based monitoring are pretty similar to that obtained by random monitoring with a monitoring coverage of 100%. This observation indicates that PCP-SBS based monitoring requires only a monitoring coverage of approximately 50% for the detection and prediction of all the runtime anomalies that directly impact the response time of OnlineLive.

Figure 5 compares the improvement in the response time of OnlineLive obtained by random monitoring and PCP-SBS based monitoring. OnlineLive beats random monitoring significantly. On average, the response time improvement obtained by PCP-SBS based monitoring is larger than random monitoring by 55.67% across all experimental cases. That indicates, given the same monitoring coverage (i.e., available monitoring resources), PCP-SBS based monitoring is on average 55.67% more cost-effective than random monitoring. Furthermore, as indicated by the improvement difference between random monitoring and PCP-SBS based monitoring in Figure 5, PCP-SBS based monitoring outperforms random monitoring by relatively large margins when the monitoring coverage is between 10% and 50%. Specifically, PCP-SBS based monitoring obtains a larger improvement in the response time of OnlineLive than random monitoring by an average of 114.81% when the monitoring coverage is 10%, 98.69% when 20%, 111.86% when 30%, 116.52% when 40% and 84.13% when 50%, much higher than the average improvement across all experimental cases (55.67%). On the other hand, when the monitoring coverage is close to 100%, there is little or no difference in response time improvement between random monitoring and PCP-SBS
the more crucial and necessary it is to focus monitoring on relatively low. We conclude that the less monitoring budget, monitoring coverage (i.e., monitoring resources) is based monitoring. This observation shows that PCP-SBS took roughly 12.2 seconds to finish the dominance calculation, criticality calculation and execution path enumeration.

VI. RELATED WORK

In recent years, the area of QoS-aware service composition has attracted much attention. Many efforts have been devoted to addressing the issue of selecting appropriate component services to fulfil the quality requirement for the SBS [10, 13]. Service Level Agreements (SLAs) are often used to provide contractual QoS guarantee for composite services [18]. At runtime, adaptation approaches are employed to handle anomalies to ensure that SLAs are fulfilled [3, 5]. However, the premise for adaptation is that the SBSs must be monitored for timely detection and prediction of runtime anomalies [3, 19].

In [20] the authors propose a framework for runtime requirement verification for service compositions. The framework supports monitoring of the component services’ runtime behaviours. Monitoring is realised by intercepting the events exchanged between the composition process and the component services. In [19] the authors propose an approach to implement runtime monitoring of WS-BPEL processes. External monitoring rules can be weaved into the service composition. In [21], an assertion language named ALBERT is proposed to specify both functional and non-functional properties of Web service compositions. At runtime, the assertions are checked by Dynamo [22], a proxy-based monitoring infrastructure. Astro, a monitoring solution, is proposed in [23], aiming at separating the business logic of a Web service from its monitoring functionality. Astro can monitor both a single composite service and multiple composite services in a class. Combing Dynamo [22] and Astro [23], the authors propose an integrated solution for monitoring service compositions in [24]. In the integrated monitoring solution, monitoring constraints can be defined on single and multiple instances, on punctual properties and on complete behaviours. In [7], SECMOL, a general monitoring language is proposed based on three existing monitoring languages, namely EC-Assertion [25], SLANG [26] and WSCoL [19]. In SECMOL, the Data Collector captures and extracts the data needed to conduct monitoring, the Data Analyzers analyses the data collected by Data Collector and the Monitoring Manager integrates and oversees the whole monitoring process. WSCoL is also adopted in [3] as a means to enrich service.
compositions with monitoring capabilities. However, none of the existing work has considered the issue of monitoring cost, i.e., the cost of conducting the monitoring, and how to cost-effectively utilise monitoring resources.

Cost-effective monitoring requires that the critical path of a SBS be identified and given priority in monitor allocation as any delays on the critical path will directly impact the response time of the SBS. However, the issue of critical path identification in service composition has not been properly addressed, particularly in response to the dynamic and volatile nature of the operating environments. In order to address this issue, we have proposed PCP-SBS, a novel approach to the identification of probabilistic critical path for SBSs, which captures the probabilistic nature of volatile operating environments. Based on PCP-SBS, cost-effective monitoring can be conducted to facilitate response time management in SBSs.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented PCP-SBS, a novel approach to the identification of probabilistic critical path for service-based systems (SBSs). A timing model is proposed which captures the probabilistic nature of the timing properties of the basic components (BCs) of a SBS. Based on the timing model, the criticalities of the execution paths and the BCs can be calculated. With the knowledge of criticalities, monitoring resources can be cost-effectively utilised. Experimental results have demonstrated that PCP-SBS based monitoring is much more cost-effective than random monitoring for response time management in SBSs. Runtime anomalies that directly impact the response time of the SBSs can be detected and predicted timely by PCP-SBS based monitoring using significantly less monitoring resources (55.67% less on average) compared to random monitoring. In addition, PCP-SBS demonstrates a slow increase in computational overhead against the number of BCs, which indicates high efficiency and scalability.

In future work, we are going to enrich PCP-SBS by extending the set of supported timing models, e.g., Markov model and queuing network model. Then, we are going to evaluate PCP-SBS in real-world large-scale operating environments.

ACKNOWLEDGMENT

This work is partly funded by the Australian Research Council in collaboration with CA Labs.

REFERENCES


