Dynamic Self-Healing Mechanism for Transactional Business Process

Yuhai Zhao and Ying Yin

College of Information Science and Engineering, Northeastern University, Wenhua Road No. 3, Shenyang, Liaoning 110819, China

Correspondence should be addressed to Yuhai Zhao; zhaoyuhai@ise.neu.edu.cn

Received 19 July 2014; Revised 27 November 2014; Accepted 18 December 2014

Academic Editor: Ashraf M. Zenkour

Copyright © 2015 Y. Zhao and Y. Yin. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

It is clear that transactional behavior consistency is a prerequisite and basis for construction of a reliable services-based business application. However, in previous works, maintaining transactional consistency during exception handling was ignored. Maintaining transactional consistency requires functionality for rolling back some operations and revoking uploaded data. Replacing only the failed service will eventually lead to overall business application failure. In this study, we take fully into account the behavioral consistency of transactional services and propose two effective self-healing mechanisms for service-based applications. If a service enters into potential failure condition, a rescheduling mechanism is triggered to maintain consistent transactional behavior and to ensure reliable execution; if a service fails during execution, the compensation operation is triggered and the system will take action to ensure transactional behavior consistency. Meanwhile, cost-benefit analysis with compensation support is proposed to minimize the dynamic reselection cost. Finally, the experimental analysis shows that the proposed strategies can effectively guarantee the reliability of Web-based applications system.

1. Introduction

In the prevalence of cloud computing and service computing, with more increasing of complexity of software and software environments, service-based business applications are facing many challenges. This is especially true when one is running composite services, where reliability is an important research topic. This paper focuses on reliable execution on transactional Web services while maintaining consistent transactional behavior to accomplish dynamic self-healing for composite service [1, 2].

A service-based business application system mainly focuses on two goals: reliability and profitability. In service-based business applications, when a customer requests a service from a provider, the Server Level Agreement (SLA) is negotiated and a contract is drawn up. The SLA stipulates the quality requirements, such as Quality of Service (QoS) and the transactional behavior relating to service to be provided and how much the customer should pay to the service for the usage of this service. Thus, as a service-based Web application, it needs to use other services (i.e., partners) to complete its advertised functionality. If the QoS is stipulated in the SLA, both customer and provider will obtain the largest profits.

However, it is well known that composite services live in a highly dynamic and failure prone Internet environment [3, 4]. Under the conditions, successful execution of a service cannot be guaranteed. There are many potential points of failure such as the deviation from normal of the quality of a single service or other exceptions that may suddenly occur in an unreliable Web service. Even seemingly small changes may undermine allowable compensation time and may therefore disrupt normal transactional behavior [5–8]; in these situations, however, operations may have been submitted partially while others may not have been submitted at all. Such exceptions can seriously disrupt data consistency of the transactional service. Partially failed transactions will lead to the overall failure of the business application. The correct handling of any exception includes not only rolling back the earlier successful operations of the composite Web service but also reexecuting a series of operations as a whole on the reselected component services.

Based on the above, it is obvious that several problems emerge during execution of a transaction. How to quickly...
respond to any raised exception in an appropriate manner and with minimal expenses is the most important thing for maintaining transactional behavior consistency between services, and further guaranteeing the reliability of these services. Therefore, maintaining consistent transactional behavior is a key problem, which cannot be ignored. This poses a challenging problem in service-based business applications.

Composite service self-healing in this paper refers to (1) rescheduling a business how to ensure consistent transactional behavior by dynamically adjusting service execution sequences; (2) replacing failed service(s) with other(s) to ensure the quality of composite service without affecting the transactional behavior. Specifically, we propose two self-healing mechanisms, which guarantee the consistency of transactional behavior by rescheduling the service execution sequence to remove potential dangers or replacing a failed subpath with new services to ensure execution reliability.

Our contributions are given below: Firstly, we propose a rescheduling algorithm which ensures composite service to enter into a safe status while avoiding potential risks. Secondly, a probability model is proposed, by which the services with minimum replace cost can be chosen in reselection. Finally, a series of experiments show that the model not only guarantees business process integrality and services with minimum replace cost can be chosen in reselection.

The rest of this paper is organized as follows. In Section 2, two examples on maintaining transactional behavior consistency are introduced. Section 3 provides a model for reliability evaluation, and presents two effective self-healing methods. Experimental analysis is given in Section 4. Section 5 describes some related work. Finally, Section 6 concludes the paper.

2. Transactional Business Process

In this section, we introduce related concepts and properties of transactional Web service. Furthermore, we show two scenarios and the corresponding challenges. Finally, we give the problem definition.

2.1. Basic Definition

Definition 1 (component service). A component service can be described by four tuples: \( s = \langle F, ET, CT, C(t) \rangle \), where \( F \) denotes the function of the Web service \( s \), denoted as \( F = \{ \text{op}_1, \text{op}_2, \ldots, \text{op}_n \} \). In this description, \( \text{op} \) denotes the \( i \) th operation of function \( F \). \( ET \) denotes the expected execution time of Web service \( s \), \( CT \) denotes the compensation time of Web service \( s \), and \( C(t) \) denote the cost of compensation of Web service \( s \) at time point \( t \).

Definition 2 (atomic). A component service is called atomic if its elements can be treated as a unit of work. That is, it can use compensation mechanisms to ensure that all of its component services complete successfully or none of them do.

For example, given a \( s = \{ \text{op}_1, \text{op}_2, \ldots, \text{op}_n \} \), all the elements of \( s \) only perform one action; either all of its components are completed successfully or do nothing by compensating to maintain the atomic properties. We say Web service \( s \) is a transactional Web service if \( s \) has one of the following transactional properties. TwS is an abbreviation of transactional Web service.

Property 1. The main transactional properties of a Web service are as follows.

1. Retriable: a service \( s \) is said to be retrievable if it is sure to be completed after several finite activations.
2. Compensatable: a service \( s \) is said to be compensatable if it offers compensation policies to semantically undo its effects.
3. Pivot: a service \( s \) is said to be pivot if once it successfully completes, its effects remain permanent and cannot be semantically undone.

These properties show compensation is the basic property for transactional Web service. A composite service or service-based business application takes advantage of transactional Web service behavior properties to specify mechanisms for recovery or failure handling. The goal of this paper is ensuring composite service reliable execution based on the above properties of transactional Web services. Based on the above transactional Web service definition and related properties, we can introduce a business process model.

A transactional business process model can be modelled in the form of \( \text{TBP} = \{ \text{TCS}, E, s, \Sigma CR, \Sigma DR, \Sigma BR \} \), where

1. transactional composite service, \( \text{TCS} = \{ \text{tws}_1, \text{tws}_2, \ldots, \text{tws}_n \} \) represents a task (transactional Web service) in the business process; a task is implemented by a series of transactional Web service operations;
2. \( E \) is a set of directed edges; \( E_j = (E_j, E_i) \in E \) corresponds to the control dependency relation between \( \text{tws}_1 \) and \( \text{tws}_2 \), where \( \{ \text{tws}_1, \text{tws}_2 \} \) ∈ \( \text{TCS} \);
3. \( \Sigma CR \) is a set of control relations for the tasks, \( \Sigma CR = \{ \text{Sequential, And-Join, And-Split, Or-Join, Or-Split} \} \), where \( \text{Sequence} \) denotes the sequence relation between tasks, \( \text{And-Join} \) denotes the parallel and unite relation between tasks, \( \text{And-Split} \) denotes the parallel and separate relation between tasks, \( \text{Or-Join} \) denotes the selective and unite relation between tasks, and \( \text{Or-Split} \) denotes the selective and separate relation between tasks;
4. \( \Sigma DR \) is set of data relations (DR) between the tasks, \( \Sigma DR = \{ 0, 1 \} \), where 0 and 1 represent the absence and presence, respectively, of data relations between tasks;
5. \( \Sigma BR \) is the set of business relations (BR) between tasks, \( \Sigma BR = \{ 0, 1 \} \), where 0 and 1 represent the absence and presence of business relations between tasks;
6. \( s : t \rightarrow \text{state} \) is a mapping function, where state = \{ initial, active, failed, completed, aborted, canceled \}. It
is easy to see that the starting state of all tasks is set to \textit{initial} before execution.

2.2. Transactional Dependency Relationship. Given a business process, we need to identify the border of transactional service once the business process bands the concrete services. Due to the autonomy of services, the transactional granularity of a composite service is hidden. Most of the previous works on obtaining the transactional granularity in exception handling are determined during the design stage. However, the obtained method is incomplete, it is different to obtain the granularity of composite introservices and nested business granularity. In addition, relying on designers to specify the business granularity is not reliable. Because Web services depend on the Internet environment which is inherently unstable, the business processes may need to run for a very long time to perform the complex business logic. Based on the multirelations, such as data dependence relation, control dependence relation, and business dependence relation in services, which make the transactional dependence relation was not dominated by designer in advance. In order to realize the composite services dynamic adaptation and to support the transactions better, we need to identify the transactional boundary and composite service granularity dynamically.

From the structural aspect, we can classify the execution scenarios into five types, namely, “Sequential” tasks, “And-Join” tasks, “And-Split” tasks, “Or-Join” tasks, and “Or-Split” tasks. According to these scenarios, if task \( t_{ws_i} \) and \( t_{ws_j} \) satisfy the following conditions, we can obtain the direct compensation dependency between tasks as defined below.

\textbf{Definition 3} (direct compensation dependency). If one task \( t_{ws_i} \) has failed and needs to be compensated, based on the state of that task and multirelation in tasks, the compensation of one task may lead to the compensation of each participant partial task. From the structural aspect, we can classify the compensation scenarios into five types, namely, Sequential tasks, And-Join tasks, And-Split tasks, Or-Join tasks, and Or-Split tasks. According to these scenarios, if task \( t_{ws_i} \) and \( t_{ws_j} \) satisfy the following conditions, we can export the direct compensation dependency (DCD) between tasks as follows:

(1) when \( t_{ws_i} \) is the direct preceding task of \( t_{ws_j} \), that is, \( CR(t_{ws_i}, t_{ws_j}) = \text{"sequential"} \), if they satisfy \( BR(t_{ws_i}, t_{ws_j}) = 1 \) or \( DR(t_{ws_i}, t_{ws_j}) = 1 \), we say there exists a compensation dependency relation, \( t_{ws_i} \prec_{\text{compensation}} t_{ws_j} \); that is, \( DCD(t_{ws_i}, t_{ws_j}) = 1 \);

(2) when two services (\( t_{ws_i} \) and \( t_{ws_j} \)) both completed before activating another service, that is, \( CR(t_{ws_i}, t_{ws_j}) = \text{"And-Join"} \), if they satisfy \( BR(t_{ws_i}, t_{ws_j}) = 1 \), we say there exists a compensation dependency relation, \( t_{ws_i} \prec_{\text{compensation}} t_{ws_j} \); that is, \( DCD(t_{ws_i}, t_{ws_j}) = 1 \);

(3) when a Web service is activated only when both of its predecessor Web services (\( t_{ws_i} \) and \( t_{ws_j} \)) have completed, that is, \( CR(t_{ws_i}, t_{ws_j}) = \text{"And-Split"} \), if they satisfy \( BR(t_{ws_i}, t_{ws_j}) = 1 \) and \( s(t_{ws_j}) = \text{"completed"} \), we say there exists a compensation dependency relation, \( t_{ws_j} \prec_{\text{compensation}} t_{ws_i} \); that is, \( DCD(t_{ws_i}, t_{ws_j}) = 1 \);

(4) when only one task will be selected from an Or-Joint multitasks or an Or-Split multitasks, that is, \( CR(t_{ws_i}, t_{ws_j}) = \text{"Or-join or Or-Split"} \), we say there exists no compensation dependency relation; that is, \( DCD(t_{ws_i}, t_{ws_j}) = 0 \).

However, in a dynamic composite environment where several component transactional Web services interact, unexpected behavior from a component Web service may not only lead to its failure, but also may bring cascade failure on the partial participants to the composition. So, determining the affected range of the cascade failure services is the first and most important step for replacement correctly. Based on the cascade range we can then select a replacement service with minimal cost. However, the affected ranges by the failure service are those services that have an indirect compensation dependency relationship with the failure service node. Indirect compensation dependency relation can be discovered by direct compensation dependencies among services.

According to different scenarios from structural analysis, we induce the indirect compensation dependency (ICD) as follows in Figure 1.

(1) \textit{Sequential} as case 1 in Figure 1(a) shows if \( DCD(t_{ws_i}, t_{ws_j}) = 1 \) and \( DCD(t_{ws_j}, t_{ws_k}) = 1 \), when a \textit{Sequential} task, \( t_{ws_k} \), is aborted or compensated, for \( (t_{ws_i} \prec_{\text{compensation}} t_{ws_j}) \land (t_{ws_j} \prec_{\text{compensation}} t_{ws_k}) \Rightarrow t_{ws_i} \prec_{\text{compensation}} t_{ws_k}, t_{ws_i} \) should be compensated; that is, \( ICD(t_{ws_i}, t_{ws_k}) = 1 \).

(2) \textit{And-Joint} as case 2 in Figure 1(b) shows if \( DCD(t_{ws_i}, t_{ws_j}) = 1 \) and \( DCD(t_{ws_j}, t_{ws_k}) = 1 \), when task \( t_{ws_k} \) is aborted or compensated, for \( (t_{ws_i} \prec_{\text{compensation}} t_{ws_j}) \land (t_{ws_j} \prec_{\text{compensation}} t_{ws_k}) \Rightarrow t_{ws_i} \prec_{\text{compensation}} t_{ws_k}, t_{ws_i} \) should be compensated; that is, \( ICD(t_{ws_i}, t_{ws_k}) = 1 \).

(3) \textit{And-Split} as case 3 in Figure 1(c) shows if \( DCD(t_{ws_i}, t_{ws_j}) = 1 \) and \( DCD(t_{ws_j}, t_{ws_k}) = 1 \), when task \( t_{ws_k} \) is aborted or compensated,

(a) when \( s(t_{ws_i}) = \text{"completed"} \), for \( (t_{ws_k} \prec_{\text{compensation}} t_{ws_i}) \land (t_{ws_i} \prec_{\text{compensation}} t_{ws_j}) \Rightarrow t_{ws_k} \prec_{\text{compensation}} t_{ws_j}, t_{ws_i} \) should be compensated; that is, \( ICD(t_{ws_i}, t_{ws_k}) = 1 \);

(b) when \( s(t_{ws_i}) = \text{"initial"} \), no compensation is needed.

(4) \textit{Or-Joint} and \textit{Or-Split} as case 4 in Figures 1(d) and 1(e) shows, for \textit{Or-Joint} tasks and \textit{Or-Split} tasks, their preceding tasks and succeeding tasks will be specific. As case 4 in Figures 1(d) and 1(e) shows, task \( t_{ws_k} \) and one of the preceding (succeeding) tasks were executed while others are not. Therefore, we can treat them as sequential tasks.

2.3. Scenarios and Challenges. In this subsection, we begin by using a simple example to show the execution process of
a service-based business application. The scenario as given in Figure 2 shows a service-based business process; each task can be implemented by invoking a set of services which span over a single or multiple Web service operations. For simplicity, we assume that one task ($tw_{si}$) corresponds to one transactional Web service modeled in Web Service Definition Language (WSDL) format. In one WSDL document, several port types are defined, each of which acts as a static interface of this Web service. A port type is composed of multiple operations, which are described in the form of the input or the output of messages. The execution of a business process task can be turned into WSDL operation invocations.

The composite service starts when it receives a request from a customer. It searches for favorite attractions first and the attraction service will recommend some popular touristic cities according to the customer's preferences. After the destination city has been determined, the composite service invokes two Web services simultaneously: a Ticket Booking service reserves an appropriate flight while the Hotel Booking service reserves an appropriate hotel. After the flight reservation and hotel reservation have been done, the composite service sends a request to the hotel service and waits for a confirmation. Upon receiving the responses from both the flight service and the hotel service, the composite service will invoke the computation service to compute the distance between the hotel and the attraction. According to the result, either the bike service or the car service with motel service is started to make the appropriate reservation. Finally, the composite service will send to the customer an arrangement in detail. The execution process of these services...
may lead to a number of possible outcomes; two of these scenarios are discussed below.

**Scenario 1** (composite service falls into potential failure). In a service-based business application, a transactional Web service has its compensational time constraint; that is, a Web service can be compensated within a specified time period. If the execution time is longer than the allotted time, there will be no compensation.

At first, we introduce a potential failure scenario as shown in Figure 3. Given two parallel executing tasks \( s_1 \) and \( s_2 \) which belong to one transactional service and are circled by a circle as shown in Figure 3(a). They both start at the same time and execute concurrently. The execution times of them are 8 seconds (ET\((s_1) = 8 \) s) and 5 seconds (ET\((s_2) = 5 \) s), respectively, while their compensation times are both 5 seconds (CT\((s_1) = CT(s_2) = 5 \) s) which are shown in Figure 3(a). During execution, if the service \( s_1 \) encounters an exception, such as traffic congestion, which causes service quality to deviate from normal values and beyond a specified threshold, it falls into risk. In this circumstance, after 4-second delay, \( s_1 \) will spend 12 seconds to finish the work.

Task \( s_2 \) on the other hand has been executed and submitted successfully. However, due to the concurrent execution of \( s_1 \) and \( s_2 \), only when both of them are completed, the next component \( (s_3) \) can be executed. Due to the transactional Web service ACID properties, component \( s_2 \) has missed the compensable time period when \( s_1 \) is completed, as shown in Figure 3(b); that is, the detention of \( s_1 \) destroys the compensation of \( s_2 \) and, furthermore, destroys the whole business process compensability and deduces the composite service into potential risk. However, the composite service can avoid potential risk if we execute service \( s_2 \) for 7 seconds delay. The process is shown in Figure 3(c).

**Scenario 2** (composite service falls into failure completely). Figure 4 will introduce another scenario: the composite service falls into complete fail phase. For example, a service operation in one task is unavailable when it is carried out half. As shown in Figure 4, payment operation \((op_3)\) of the Booking Ticket process \((tws_1)\), that is, the red dots, fails when it is carried out half.

One solution is to replace the failed service \( tws_1 \) with another. However, at the same time, query operation \((op_1)\)
and booking operation (op\textsubscript{2}) in tws\textsubscript{1} have been executed and submitted successfully. That is, the result (i.e., the ticket has been selected by a specific user and other customers will not be able to inquire this ticket information) of booking operation (op\textsubscript{2}) remains resident in memory and will not be released. Meanwhile, the user has paid for the reservation in tws\textsubscript{2} with a certain discount due to prior agreements between the hotel and the airlines (since the hotel booking will provide 20% discount due to business relationship).

tws\textsubscript{2} and tws\textsubscript{3} belong to one transactional service. Then, according to the transactional Web service ACID properties, tws\textsubscript{2} needs to be compensated when tws\textsubscript{3} needs to be replaced. That is, the failure of one Web service (tws\textsubscript{1}) will induce another cascading service (tws\textsubscript{2}) to fail. However, replacement and compensation are costly and a long time may be required. Eventually, this will lead to a cascading compensation and replacement issue. For this reason, adopting the traditional replacement model directly, which will lead to cascading inconsistencies in the client server data, may not be the best solution in this case.

Problem Statement. Given an initial executing service sequence (SS) and SLA, the task at hand is to analyze the implementation status of services, to adapt an appropriate strategy to satisfy the SLA requirement and guarantee the system is always in safe areas or reselect candidate services to replace failed sets with minimal cost.

3. Two-Phase Framework for Self-Healing Mechanism

3.1. Basic Two-Phase Framework. The two-phase self-healing framework consists of two stages: the early prediction stage [9] and the adaptive self-healing stage. Moreover, in the self-healing stage, we propose two self-healing strategies to deal with the potential failure and complete failure, respectively. Figure 5 gives an illustration of the two-phase framework.

Stage 1 (the early prediction stage). In Figure 5, the early prediction process is illustrated by the flowchart in the left dashed-box. In this process, execution engine first invokes the services in service resource to complete a specific business process. Then, the corresponding execution information is recorded in Execution Log or Fault Log. Meanwhile, the status of the composite web service is monitored. Once the status matches a fault pattern, which is a web execution sequence mined from Execution Log and Fault Log by the early pattern mining [9], the self-healing mechanism is triggered. Note: the early pattern refers to a pattern, which (1) is frequent in the failed web execution sequences; and (2) is as short as possible and is of high prediction accuracy. The properties of the early pattern are very important for the online QoS prediction. (1) means the pattern is statistically significant in the failed web execution sequences; and (2) is as short as possible and is of high prediction accuracy. Thus, the timeliness of the early pattern based prediction of QoS is guaranteed.

Stage 2 (the self-healing stage). The self-healing process is illustrated by the flowchart in the right dashed-box. In this stage, by fault type analysis, we first decide that it is the running failure or the compensable failure. If it is the running failure, the self-healing mechanism invokes the reselection algorithm. Otherwise, if the compensable fails, we invoke the rescheduling algorithm. As soon as the adaptive strategy is conducted, the system configures the web service resources according to the new strategy, repairs the faults, and saves the newly generated service sequence to the Execution Log.

The main advantages of this model are as follows. First, trigger based on the early prediction by this method is robust. Because the early patterns mined were based on previously
Mathematical Problems in Engineering

**ALGORITHM 1: Self-healing algorithm with scheduling (SA1).**

executed service sequences, the self-healing mechanism will be triggered once the service status monitored online matches an exception service sequence. Secondly, this method can deal with different failure scenarios, such as potential failure and complete failure, which is correct and efficient.

### 3.2. Self-Healing Mechanism for the Problem on the Potential Fail by Scheduling

A business process specifies the order in which component services are invoked and the conditions under which service twsᵢ may be invoked.

**Definition 4** (transactional business process (TBP)). A transactional business process (TBP) can be viewed as an execution sequence with time constraint, such as TBP = \{⟨tws₁, tᵢ₁⟩, ⟨tws₂, tᵢ₂⟩, …, ⟨twsᵢ, tᵢ⟩\}, where twsᵢ denotes component service and tᵢ denotes the scheduled starting point of the execution of service twsᵢ.

We say twsᵢ is atomic if the elements of the service twsᵢ can be treated as a unit of work. That is, it can use compensation mechanism to ensure that all of its component services are completed successfully or none of them do; we say it is Atomic Transactional Service (ATS).

**Definition 5.** Given two parallel component services twsᵢ and twsⱼ which belong to one transactional service, we say they are in a safe region if and only if PT(twsᵢ) − CT(twsᵢ) > δ (δ is a given threshold by user); we say they are in critical region if and only if 0 < PT(twsᵢ) − CT(twsᵢ) < δ; we say they are in risk region if and only if PT(twsᵢ) − CT(twsᵢ) < 0. In this equation, PT(twsᵢ) is the prediction time of service twsᵢ; CT(twsᵢ) is the compensation time period of service twsᵢ.

**Definition 6** (optimal business process scheduling (OBPS)). We say a business process scheduling (BPS) is an optimal composite service scheduling sequence if BPS satisfies the following conditions: (1) all the component services can be allowed to compensate during the entire life period and (2) its total compensation cost is minimal at the moment.

If given the following conditions: (1) BPS, a business process scheduling, (2) tᵢ, time point. Our goal is monitoring and finding an optimal composite service scheduling which satisfies Definition 5.

We present our optimal self-healing algorithm by scheduling, called SA1 (see Algorithm 1). The SA1 algorithm has two main steps: (1) First, we mine all multirelationships between Web services and identify transactional service granularity based on the business process. As such, we can construct the initial composite service scheduling. (2) Secondly, monitor and predict the composite service quality (using the method in [10]) and determine the transactional service compensability. Once one of the component services will go into the critical region, the composite service schedule is adjusted and the compensability state is returned with the most optimal scheduling.

### 3.3. Self-Healing Mechanism for the Problem on the Completed Fail by Dynamic Reselection

In order to maintain data consistency of transactional Web service, a composite service needs compensation when a failure occurs. Furthermore, in order to guarantee the reliable execution, the system needs to reselect new service(s) to replace the affected service(s). To analyze the expected reselection cost with compensation, we define the snapshot and analyze different compensation costs furthermore as follows.

**Definition 7** (snapshot). A snapshot of the execution of a composite service at time tᵢ is a 5-tuple \{SC₁, SC₂, S_EXE, Sₛᵢ, Sⱼ\}, where tᵢ refers to the current time point and SC₁ is a set of
compensable component services that have been completed before time $t_i$, $SC_2$ is a set of component services that have been completed at time $t_i$, $SC_{EXE}$ is a set of component services that are being executed at time $t_i$, $S_S$ is a set of component services that start at time $t_i$; $SC_1$ is a set of component services that have not yet started at time $t_i$.

Because compensation operation only occurs on those services which have finished before or will finish at time $t_i$, there is no compensation cost for services that have not yet started or will start at this point in time (see Figure 6).

Based on the above equations, we compute compensation cost as follows.

1. During the execution, for a composite service, if a failure does not occur before time $t_i$, compute the successful probability $p^s_{i,j}$. The expected compensation cost depends on the finished services and those services which will finish at time $t_i$. In this case, business process (BP) runs successfully before time $t_i$; that means, service in $SC_1$ has been completed successfully, service in $SC_2$ will be complete at time $t_i$, and $SC_{EXE}$ has started successfully. So, the compensation cost can be computed by the following:

$$Q_{ct} = \sum_{j=1}^{SC_1+SC_2} p^s_{i,j} C_{s_j}^i + \sum_{j=1}^{SC_{EXE}} p^f_{i,j} C_{s_j}^i.$$  \hspace{1cm} (1)

2. During the execution, for a composite service, if a failure occurs at time $t_i$, compute the failure probability $p^f_{i,j}$. The expected compensation cost depends on the finished services ($SC_1$), those services ($SC_2$) which will finish at time $t_i$, and those ongoing services ($SC_{EXE}$). So, the compensation cost can be computed by the following:

$$Q_{ct} = \sum_{j=1}^{SC_1} p^f_{i,j} C_{s_j}^i + \sum_{j=1}^{SC_{EXE}} p^f_{i,j} C_{s_j}^i.$$  \hspace{1cm} (2)

Given Web service with compensation support, $s_j$, its compensation cost denoted by $C(s_j, t_i)$, $C(s_j, t_i) = \alpha \times Q^f_{ct}(s_j) + \beta \times Q_{ct}(s_j)$, where $Q^f_{ct}(s_j)$ denotes the cost by executing $ct_{ws_j}$ at time point $t_i$ and $Q_{ct}(s_j)$ denotes the cost by executing $ct_{ws_j}$.

Therefore, the compensation cost of a finished service equals the probability of successful implementation of service at time $t_i$ multiplied by the compensation cost.

### 3.3.1. Benefit-Cost Analysis (BC-A) for Adaptive Service Reselection

As opposed to previous pure replacement algorithms, we proposed a comprehensive, objective, and effective self-healing model. Our self-healing model not only provides transaction support but also ensures the optimization of reselected service QoS and flexible compensation cost.

$$\text{Score}^f(s_j) = UF^f(cs_j) + UF(s_j) = \sum Q^f_{ct}(cs_j) + \sum Q^f_{ct}(s_j),$$  \hspace{1cm} (3)

where $UF(cs_j)$ denotes the utility function of compensation service $s_j$ and $UF(s_j)$ denotes the utility function of reselective service cost. $\sum Q^f_{ct}(cs_j)$ denotes the total of quality description of compensation service at time point $t_i$ and $\sum Q^f_{ct}(s_j)$ denotes quality description of selection by picking up $k$th path. $m, n$ represent number of compensation service parameters and replacement service parameters, respectively. $r$ denotes the length of rollback. $r + x$ denotes the length of reselection service sequence. Further, based on the QoS criteria, we obtain the detailed formula as shown below:

$$UF^f(ct_{ws_j}) = \sum_{t=1}^{TWS} Q^f_{ct}^{(t)}(CTWS)$$

$$UF^f(t_{ws_j}) = \sum_{t=1}^{TWS} Q^f_{ct}^{(t)}(TWS)$$

$\beta \in \{\text{price, time}\}$, $\gamma \in \{\text{available, prefer, successful, ...}\}$. The QoS criteria of compensation service includes price and time; therefore, the detailed compute process is shown in formula (2) and (3). However, for general services, its QoS criteria are different; some of the criteria used could be negative; that is, the higher the value is, the lower the quality is. This includes criteria such as execution time and execution price. Other criteria are positive criteria; that is, the higher the value is, the higher the quality is. In this paper, $W_a$ and $W_b$ are the weight assigned to negative quality criteria and positive quality criteria, respectively. In order to balance or normalize the criteria, values are scaled according to (4) for negative criteria; values are scaled according to (5) for positive criteria:

$$V_x(t_{ws_j}) = \begin{cases} \frac{QoS^x_{max}(t_{ws_j}) - QoS^x_{min}(t_{ws_j})}{QoS^x_{max}(t_{ws_j}) - QoS^x_{min}(t_{ws_j})} & \text{if } QoS^x_{max}(t_{ws_j}) - QoS^x_{min}(t_{ws_j}) \neq 0, \\ 0 & \text{otherwise} \end{cases}$$

$$V_x(t_{ws_j}) = \begin{cases} \frac{QoS^x_{max}(t_{ws_j}) - QoS^x_{min}(t_{ws_j})}{QoS^x_{max}(t_{ws_j}) - QoS^x_{min}(t_{ws_j})} & \text{if } QoS^x_{max}(t_{ws_j}) - QoS^x_{min}(t_{ws_j}) \neq 0, \\ 0 & \text{otherwise} \end{cases}$$

Figure 6: An execution snapshot of time point $t_i$. 
In (6), \(QoS_{\text{max}}^x\) is maximum value of a quality criterion for \(x\); that is, \(QoS_{\text{max}}^x = \text{Max}(QoS_{x,i})\). \(QoS_{\text{min}}^x\) is minimum value of a quality criterion for \(x\); that is, \(QoS_{\text{min}}^x = \text{Min}(QoS_{x,i})\).

Further, we got the detailed replacement cost at time point \(t\) for selecting the \(k\)th path as shown

\[
\text{Score}_k (\text{tws}_j) = \text{Max} \left\{ \sum_{\alpha=1}^{m} W_{\alpha} V_{\alpha} (\text{CTWS}) + \sum_{\beta=1}^{n} W_{\beta} V_{\beta} (\text{TWS}) \right\} = \begin{cases} \\
\sum_{\alpha=1}^{m} W_{\alpha} + \sum_{j=1}^{\rho_1} \frac{W_{\beta}}{3} + \sum_{j=1}^{\rho_1} \frac{V_{\beta}}{3} \left( \text{tws}_j \right) ^k \\
\sum_{\alpha=1}^{m} W_{\alpha} \left( \text{tws}_j \right) \leq \text{SLA}^x; \\
\sum_{\beta=1}^{n} V_{\beta} \left( \text{tws}_j \right) \leq \text{SLA}^\beta,
\end{cases}
\]

where \(\text{SLA}^x\) is the max compensation cost for attribute \(W_{\alpha}\) and \(\text{SLA}^\beta\) is the max reselection cost for attribute \(W_{\beta}\).

### 3.3.2. Optimal Service Reselection Algorithm.

The self-healing ability is an important feature in adaptive systems. Good self-healing mechanism includes not only repairing by itself but also executing with minimal cost and minimal interrupt time delay. Based on the idea, unlike the previous purely direct replacement strategies, this paper fully considers transaction properties and proposes a self-healing algorithm with compensation support and reselection support. Furthermore, we give an optimal service reselection algorithm which considers cost profit analysis.

The running failure oriented self-healing algorithm (SA2) includes three main steps: (1) Find the unavailable node and judge whether or not it should be compensated; (2) if the node needs to be compensated, mine the minimal compensation scope affected by the unavailable node based on multirelations, such as control relation, data relation, and business relation; (3) finally, induce the minimal scope of replacement by matching behavior interface [II] and reselect the optimal replacement services by benefit-cost analysis; finally, we show the SA2 algorithm (see Algorithm 2).

We give the outline of the algorithm followed by the discussions on every main step. First, label the nodes and edges connected to the failure node \(\text{tws}_j\) (lines 2-3); if the node needs to be compensated, automatically search the minimal compensation scope affected by the unavailable node based on existing multirelations (lines 4–9) such as control relation, data relation, and business relation; secondly, mine the matching behavior interface with minimal length and replace it (lines 10–13); the detailed process refers to [II]. Limited by space, we do not explain the function in detail. Finally, reselect the optimal replacement services by benefit-cost analysis (lines 14-15).

### 4. Experiments

The following experiments mainly analyze the efficiency and the success rate of the proposed self-healing composite service model. For brevity, we refer to the self-healing algorithm with scheduling as SA1 and the self-healing algorithm without scheduling as NSA1. We simulate the network environment and generate the network topology graph by BRITE tool. The number of web services varies from 40 to 240. Specifically, these services are divided into 3 to 10 classes. The execution period is 10 weeks. The system selects the composite services by frequency.

The first series of experiments aim to compare the performance of SA1 and NSA1. Figure 7 shows the average success rate of SA1 and NSA1 under different periods (from one week to six weeks), where 100 different services make up 15655 distinct service execution sequences. Figure 8 shows the average success rate of SA1 and NSA1 under different tasks (from 3 tasks to 8 tasks), while the fault rate is 5% and the running period is 6 weeks. Figure 9 shows the average success rate of SA1 and NSA1 under different fault rates (from 1% to 6%), where 100 different services make up 15000 distinct service execution sequences. As we can see, the success rate of SA1 is better than that of NSA1. This illustrates that self-healing algorithm with compensational scheduling is more robust and reliable. When rescheduling the potential failure service, the system will escape from risk and be in security.

The second series of experiments aim to compare the performance of SA2 and Yu's method [12]. Figure 10 shows the average success rate of the two algorithms for SA2 and Yu under different periods (from one week to six weeks), when the number of services is 100 and consists of about 15655 tuples. Figure 11 shows the average success rate of the two algorithms for SA2 and Yu under different datasets (from 40 components to 240 components), when the number of services is 100 and consists of about 15655 tuples. Figure 12 shows the average success rate of the two algorithms for SA2 and Yu under different fault rates (from 1% to 6%), when the number of services is 100 and consists of about 15655 tuples. As we can see that the success of SA2 is better than that of SA1 and NSA1. This illustrates that self-healing algorithm with compensation support is more robust and reliable. When the failure service needs to be compensated, the applicability of existing pure replacement algorithm is poor.

The third series of experiments are conducted to evaluate the scalability of the proposed method. Figure 13 shows the scalability for SA2 under different lengths of rollback, when the number of services is fixed to 50 and 100. That is, when the value of the parameter (represented by the \(x\)-axis) increases, the run times of SA2 (represented by the \(y\)-axis) go up.
Algorithm 2: Self-healing algorithm with compensation (SA2).

The shorter the rollback length, the lower the run times showing an approximately linear relation. Figures 14 and 15 show the average running times of SA2 and Yu’s method under different running periods and different tasks. As we can see, Yu’s method is better than our method when the amount of data is small. However, when the data or the number of tasks is accumulated to a certain time, proposed algorithm (SA2) performs better than Yu’s method. We can see that the success rate of SA2 is higher than the existing purely replacement algorithm, it illustrates that the self-healing algorithm with compensation support is more robust and reliable. When the failed service needs to be compensated, the applicability of existing pure replacement algorithm is poor.

The fourth series of experiments aim to analyze the overhead of the proposed two-phase framework, including time taken in the early detection (TD) and in the self-healing (TS) for SA1 and SA2, respectively. The experiments conducted for SA1 are shown in Figures 16–18. Figure 16 shows the overhead of SA1 under task = 6 and task = 8 while period varies from 1 to 6 and fault rate is fixed to 3%. Figure 17 shows the overhead of SA1 under period = 3 and period = 5 while fault rate varies from 1% to 6% and task is fixed to 6.
Figure 9 shows the success rate versus fault rate. Figure 10 shows the success rate versus period. Figure 11 shows the success rate versus task. Figure 12 shows the success rate versus fault rate.

Figure 18 shows the overhead of SA1 under fault rate = 3% and fault rate = 5% while task varies from 4 to 9 and period is fixed to 3. As seen from the figures, time taken in the early detection is much shorter than that taken in the self-healing. This is because the early patterns are guaranteed to be the sequences of as short as possible size but as high as possible prediction accuracy [9]. Thus, the early detection time is short. Note: we do not count time taken for mining the early patterns since they can be mined offline before triggering the prediction of QoS. The experiments conducted for SA2 are shown in Figures 19–21, where different number of services corresponds to different cases. Specifically, the cases of 50 and 100 services are, respectively, referred to as case 1 and case 2. Figure 19 shows the overhead of SA2 under case 1 and case 2 while length of rollback varies from 2 to 7 and task is fixed to 8. Figure 20 shows the overhead of SA2 under task = 6 and task = 8 while case varies from 30 to 180 and length of rollback is fixed to 4. Figure 21 shows the overhead of SA2 under length of rollback = 3 and length of rollback = 5 while task varies from 4 to 9 and case is fixed to case 1. Similar to the figures for SA1, time taken in the early detection for SA2 is also much shorter than that taken in the self-healing. This
The length of rollback
Run times (ms)

Case 1
Case 2

Figure 13: The scalability for SA2.

Run times (ms)

SA2
Yu

Figure 14: Running time versus scenarios.

Run times (ms)

TD
TS

Figure 15: Running time versus tasks.

Run times (ms)

Task

Figure 16: TD and TS versus period.

is also because the small sizes of the early patterns lead to the short time of the early detection.

Further, we give the compensation cost analysis in Table 1. Totally, 10 datasets are used, where the number of services (column 2) range from 20 to 200, and the maximum length of the behavior interface matching (column 3) ranges from 5 to 10. For each row in column 3, we further define the corresponding maximum compensation length (column 4). With the compensation cost randomly set between 0 and 1, columns 5 and 6 are, respectively, the average compensation time and the average compensation cost. As seen from Table 1, the values in columns 5 and 6 hardly vary with the number of rollback compensation services increasing. This indicates that our method is of nice scalability.

5. Related Work

With the rapidly increasing complexity of systems, how to ensure the composite web services reliably executed without interrupted by exceptions is one of the most challenging problems. Reliability execution refers to composite services that can identify unavailable services and reselect new services to replace the unavailable web services with the dynamically changing environment. This kind of adaptive
mechanisms guarantees the business process will not be interrupted and can be executed reliability and therefore attracts much attention from academics and industry.

Substitution is one of the most important mechanisms that guarantee the system reliability. There are two classes of substitution mechanisms at present. The first substitution strategy is replacement oriented service function [13–15]. For example, based on the idea of the replacement composite service, in [15], the authors mention services with the same parameters can provide similar functions; they discover services by matching similar parameters and semantic function. Reference [14] proposes a service replication approach, in order to substitute the original component service when it is not available due to the traffic congestion. Based on the idea of replication, [13] proposes a service composition approach based on redundancy mechanisms. The key to this approach is to establish a set of redundant services for each component service. Then, if one component service fails, the service can be replaced with an alternative member of the same redundancy group. Another substitution strategy is replacement oriented quality (i.e., QoS) [12, 16–18]. For example, based on the idea of the replacement composite service, some researchers [12] propose approaches of backing up a composite service for each component service. Then, when a component service failure occurs, the composite service can easily switch to a replacement one and such self-healing process will not cause an extra delay. In [12, 17], all the replacement composite services are backed up before the execution of the composite service. Such two approaches do not consider the QoS in the execution of the composite service. Because of the dynamic nature of Web services, the replacement service may not be available at all times. The approaches in [16, 18] are two studies on reselection in the execution of the composite service. In [16], the author proposed composite service replacement algorithm for global optimization. The method focuses on
reselecting the unexecuted services when the failure was triggered and ensuring the global QoS as soon as possible. In [18], the reselection will be triggered as soon as the actual QoS deviates from the initial estimates. When the failure is found, the execution of the composite service will be stopped until the reselection is completed. All in all, they only analyze the QoS requirement for replacement, without ensuring the overall system consistency due to lack of transactional support. Moreover, transactional properties can guarantee the composite service execution reliability [19]. Those replacement algorithms ignoring transaction support will fail even satisfying the requirements from function or semantical point of view. Under these circumstances, the system will be interrupted and the application was limited. Note: in one of our previous works [20], a simple replacement model of QoS was proposed, where we proposed that both the transactional replacement cost and the compensation cost should be considered. Compared with the work in [20], this paper is of two different contributions: (1) a novel rescheduling algorithm is presented, which ensures composite service to enter into a safe status while avoiding potential risks, and (2) a replacement model of QoS with probability consideration is proposed, by which the services with minimum replacement cost can be chosen in reselection.

6. Conclusion

We proposed a self-healing framework in order to make service-based application reliable execution. Firstly, we propose a rescheduling algorithm which ensures composite service enters into safe status from potential risk. Secondly, a probability model is proposed, which reselects services with minimal cost. Such an approach is an integration of flexible compensation service in rescheduling and reselecting in execution. In order to make the composite service healing itself as quickly as possible and minimize the number of reselections, a way of mining cascading scope of replacement in advance by considering fully multirelation between transactional Web services is proposed in this paper. On this basis, a new comprehensive, objective QoS-driven services reselection model with compensation supporting was described; further, the self-healing QoS-driven services reselection. Finally, A series of experiments show that the model not only guarantees business process completion and consistency, but also enhances system’s reliability and credibility.
Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by National Natural Science Foundation of China (61272182, 6100028, 61073063, 6173030, and 6173029), 863 program (2012AA010004), 973 program (2011CB302200-G), National Science Fund for Distinguished Young Scholars (61025007), State Key Program of National Natural Science Foundation of China (61332014), New Century Excellent Talents (NCET-11-0085), China Postdoctoral Science Foundation (2012T50263, 2011M500568), and Fundamental Research Funds for the Central Universities (N130504001).

References


Submit your manuscripts at http://www.hindawi.com