2013 IEEE 20th International Conference on Web Services

Execution Recovery in Transactional Composite Service

Cao Jiuxin, Zhou Tao, Zhu Gongrui, Liu Bo, Luo Junzhou
School of Computer Science and Engineering
Key Laboratory of Computer Network and Information Integration of MoE of China under Grants No. 93K-9
Southeast University, Nanjing, 211189, China
{jx.cao, zhoutao, zhugongrui, bliu,jluo}@seu.edu.cn

***Abstract*—In the composite service which runs for a long time under the heterogeneous and loose-coupled circumstance, the failure of service tends to occur. The transaction and recovery mechanism is urgently needed in order to guarantee the end-to-end QoS of the workflow and satisfy the user requirement. In this paper we address the composite service recovery issue in the way of substitution with the consideration of QoS constraint, based on our previous research work of the transactional construction and processing rules and the global optimization service selection algorithm, TSSA. Firstly, the service execution graph (SEG) is introduced and a service execution solution selection algorithm is proposed to choose one from the solution set of TSSA which has the highest success rate of recovery. Then, the concepts of execution backup path and switch cost are introduced and a search algorithm is presented to search for the optimal backup path when current service failure occurs. Meanwhile, a local induction algorithm based on positive feedback is described which could rapidly construct an transactional execution path when no backup execution path can be found in SEG and also guarantees the transactional constraint of composite service. Finally, experimental results show the recovery algorithm proposed in this paper is efficient and has high reliability.**

Keywords-composite service; service recovery; transaction process; backup path; local induction;

I. INTRODUCTION

Web Service (WS), as well as the Service-Oriented Architecture (SOA), provides effective support to the resource sharing and application interoperation under the distributed and heterogeneous environment. Increasingly more enterprise applications and business cooperations rely on WS. But normally the function of single WS is limited and can hardly satisfy the requirement of complex application. So it is needed to combine the WSs together based on the workflow described in WS-BPEL [1] and form the Composite Service (CS), in order to achieve a complicated task. Currently, CS has been adopted widely to improve the agility, flexibility and serviceability of enterprise software system.

To satisfy complex application requirement, CS organizes the WSs distributed in different autonomous domains based on certain workflow. But as the component WSs come from different providers and autonomous domains, and also because of the openness and changeability of Internet, the CS workflow may be unable to complete the execution as planned due to the failure of the WSs. A recovery mechanism is urgently needed by CS to handle the WS failure issue, in order to guarantee the end-to-end QoS of the whole workflow and satisfy the user requirement. Rahmani et al. [2] claim that there are mainly two types of service recovery approaches: forward recovery and backward recovery. Forward recovery handles the exception in certain way and enables the CS to continue execution following the workflow to accomplish the goal. In backward recovery, due to the unreachability of the goal or the unsatisfiability of the QoS constraint, the CS is compensated in certain way, and the effects caused by the executed part of business process are "erased". The approach described in this paper belongs to forward recovery.

In our earlier work, we have proposed the construction and processing rules and a global optimization service selection algorithm in ICWS2012 [3]. On the basis of them, we present a service recovery algorithm based on substitution in this paper. We consider recovery from the perspective of substitution, combined with construction and processing rules to guarantee the atomic consistency of CS. The experimental result shows the recovery algorithm proposed in this paper is efficient and has high reliability.

The remainder of the paper is organized as follows. In Section II, the related work is presented. The preliminaries including the construction and processing rules and the global optimization service selection algorithm are introduced in Section III. A service recovery algorithm based on substitution is proposed in Section IV. In Section V, the performance of our algorithm is evaluated. Finally, Section VI concludes the paper.

II. RELATED WORK

Web service composition is the key technique to implement SOA, and constructing reliable SOA application mainly depends on service recovery. Currently, many service recovery mechanisms have been proposed. Yu et al. [4] adopt redundant hardware and software resources to improve the reliability and fault tolerance of WS. FT-SOAP system proposed in [5] intercepts the user requirement at message layer. When WS fails, it will be redirected to the backup service automatically. Ye et al. [6] propose a middleware based on active replication to support reliable WS. In [7], authors introduce the WS-Replication framework to ensure high serviceability of WS and it could satisfy the system requirement to execute key tasks. This kind of techniques normally need the support of specific middleware and

978-0-7695-5025-1/13 $26.00 © 2013 IEEE
DOI 10.1109/ICWS.2013.45

require to maintain the consistency of code and state among the backup services. So the implementation process is complicated. Moreover, they are mainly used to improve the reliability of the web service and do no concern on the QoS property, so the end-to-end QoS requirement of CS could not be guaranteed. Our previous work in GCC2010 [8] utilizes the method of combining forward and backward recovery and proposes a business transaction recovery mechanism based on constraint rules. By introducing constraint rules, the recovery of transactional CS obtains higher flexibility and efficiency. Although this method is able to ensure the atomic consistency of CS, it cannot guarantee the end-to-end QoS requirement yet.

Currently, researchers have already made some studies considering QoS constraint. Canfora et al. [9] provide a method which can trigger and run a re-planned CS during execution. The assessment of QoS triggers the re-planning of CS in the process of execution. This method guarantees the QoS requirement. However, the re-planning of CS is based on the frequent reassessment of the whole workflow. This will lead to the increasing of computing cost. Yu et al. [10] propose two dynamic process re-planning algorithms on the premise of ensuring end-to-end QoS constraint. Based on the concept of backup path, the first algorithm called CSPB generates the backup path from current service to the end of the workflow. It has a high computing complexity. In the second algorithm called CSPR, when there are multiple execution paths, the global optimal reselected path is obtained by running service selection algorithm repeatedly. Obviously, the time complexity of this algorithm is great and the CS could not continue executing when the algorithm is running. Both of the two algorithms do not support processing on multiple invalid services and non-sequential workflow. To solve these problems, literature [11, 12] uses iteration algorithm to find an area with few web service to lower the computing cost of re-planning.

Unlike the methods above, this paper obtains certain number of service execution solutions at the selection phase of CS, then use the selection algorithm to choose one with the largest weight from multiple execution solutions(this one has relatively higher success rate of recovery). If invalid service occurs when executing service, first search for the optimal backup path. Backup path is constructed using service execution solutions. We will give its detailed definition in Section IV. In case no such backup path could be found, the local induction algorithm is used to construct an execution path for recovery.

III. PRELIMINARY

In our earlier work, we have proposed the construction and processing rules and a global optimization service selection algorithm known as TSSA [3]. This section, starting from some definitions about composite service and transactional properties, mainly gives a general review about our previous work, providing background of the following recovery mechanism. For more detailed explanation please refer to [3].

Definition 1. Concrete Composite Service (CCS) CCS consists of a set of component WSs which are designated from the candidate services set for each corresponding Abstract Service (AS) included in Abstract Composite Service (ACS). Therefore, one ACS can be associated with multiple CCSs.

Transaction technology is proposed to handle the problems such as data or logical inconsistency of CS caused by failed WS. Based on existing transaction model proposed in [13], we have the definitions of transactional property as following.

Definition 2. Transactional property of the individual WS Transactional property of a WS s. $TP(s) \in TPSet$, $TPSet = \{p, c, r, cr\}$, more detailed explanation of each transactional property as follows:

1) $TP(s) = p$, means the transactional property of s is pivot. Once s successfully completes, its effects cannot be semantically undone. If it fails, it has no effect at all.

2) $TP(s) = c$, means the transactional property of s is compensatable, as another WS s' exists which can semantically undo its execution effects.

3) $TP(s) = r$, means the transactional property of s is retriable, as it guarantees a successfully termination after a finite number of invocations.

4) $TP(s) = cr$, means the transactional property of s is a combination of the two transactional properties of c and r.

Definition 3. Transactional property of CS The transactional property of CS ccs, $TP(ccs) \in TPSet$, $TPSet = \{p, c, r, cr, n\}$. More detailed explanation of each transactional property as follows:

1) $TP(ccs) = p$, a ccs is pivot if once all its component WSs execute successfully, their effect remains forever and cannot be semantically undone. On the other hand, if one component WS does not execute successfully, then all previously successful component WSs have to be compensated.

2) $TP(ccs) = c$, a ccs is compensatable if all its component WSs are compensatable.

3) $TP(ccs) = r$, a ccs is retriable if all component WSs in ccs are retriable.

4) $TP(ccs) = cr$, means ccs has both the transactional properties which are defined in 1)and 2).

5) $TP(ccs) = n$, means ccs does not satisfy the conditions which defined in 1), 2), 3), 4) at the same time.

Definition 4. The atomic consistency of the CS Atomic consistency, equivalent to transactional constraints, could be satisfied if and only if a CS satisfies any of the following two conditions,

a. Each component WS in CS can be executed successfully.

b. Any of the component WSs included in the CS fails, the all previously successful executed component WSs must can be compensated.

Definition 5. Transactional CS (TCS) A CS which satisfies atomic consistency is called TCS. Definition 4 and Definition 6 show that $TP(tcs) \in \{p, c, r, cr\}$, where tcs is a concrete CS.

A. The Construction and Processing Rules of the TCS

We use workflow to describe the ACS and different workflow patterns describe the different relationship between

the ASs. Here we discuss the CS including the sequential pattern and parallel pattern. The parallel pattern is divided into two types:

Definition 6. Simple Parallel Pattern A parallel pattern which only has two branches, each of them only has one abstract service and not contained by other parallel pattern is the so called simple parallel pattern.

Definition 7. Complex Parallel Pattern In addition to the simple parallel pattern the rest part of the parallel pattern is complex parallel pattern.

Based on the above definitions, we proposed the transactional guarantee rules also known as **the construction rules and processing rules of TCS** and gave its theory proof in our paper [3].

B. Transactional Service Selection Algorithm (TSSA)

We have presented the Transactional Service Selection Algorithm (TSSA) to solve the transactional service selection issue. TSSA is based on Ant Colony System introduced in [14], wherein construction and processing rules are applied to search candidate services set and it not only can ensure the atomic consistency of the selected CS but also can obtain high efficiency. By applying TSSA, K composition near-optimum solutions could be obtained, each of which is a service execution solution.

TSSA includes two important sub algorithms, which respectively are Accessible Candidate Service Set Find Algorithm (ASSFA) and Solution Update Algorithm (SUA). ASSFA finds accessible WSs in candidate service set using both construction and processing rules of TCS. It could shrink the searching space of the algorithm tremendously to improve efficiency of the TSSA. SUA updates the container of solutions in order to store the optimal solutions, using the domination relation and QoS constraint.

IV. SERVICE RECOVERY BASED ON SUBSTITUTION ALGORITHM (SRBSA)

The recovery strategy is based on service execution solution selection, backup path searching and local induction. Firstly, the set of service execution solutions, which is the CCS set of the ACS, could be obtained by applying TSSA. With the solution set, the Service Execution Graph is constructed. Then, one execution solution which has the highest success rate of recovery is selected from the set according to the Service Execution Graph. The rest of the solutions could be used to construct backup path. If invalid service occurs, the recovery algorithm will search for the optimal backup path. Once there is no such backup path, local induction algorithm is invoked and tries to find a best execution path. If satisfies the QoS constraint, the execution path could be used to accomplish the recovery.

A. Service Execution Graph

As the service execution solution is selected according to the Service Execution Graph (SEG), first we will introduce the construction rules of SEG.

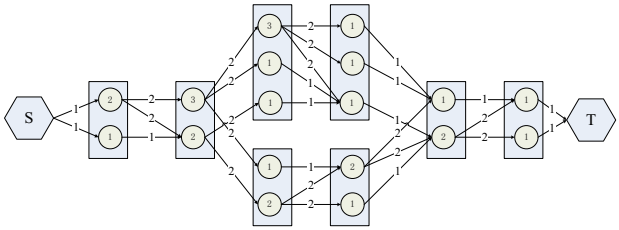

Figure 1. Service Execution Graph

Fig.1 shows an example of SEG. The edges in SEG are composed of K service execution solutions obtained by TSSA. The SEG construction rules are as following:

1. Number the ASs in ACS from left to right in sequential pattern and from top to bottom in parallel pattern. Assume there are N ASs in ACS, then we have ASs like AS_i (i = 1…N).
2. Nodes in SEG are the WSs in the candidate service sets of the ASs which appears in certain service execution solution. Assume there are Q_i WSs obtained by TSSA in AS_i (i = 1…N), then WS_{ij} (j = 1…Q_i) is a WS in AS_i, and it is a node in SEG.
3. Add an edge between two nodes of SEG if they are neighboring in a service execution solution.
4. Add virtual source node S and virtual destination node T.
5. The value marked inside the node denotes the out-degree of the node.
6. Designate the weight W of an edge as the out-degree of its head node. If the out-degree is larger than 2, let the weight be 2. It is to reduce the influence of certain nodes with large out-degree on the selection of the whole solution.

B. Service Execution Solution Selection Algorithm

K composition solutions without dominating each other are provided by TSSA. Each one is a service execution solution. Although the merits of the solutions could not be distinguished in the aspect of QoS, they may have different success rates of recovery. In this part, we will introduce a selection algorithm to choose one from K solutions as the final CCS to execute. The selected solution has the highest success rate of recovery.

The Optimal Recoverable Service Execution Solution Selection Algorithm based on SEG is presented below:

Algorithm 1: Optimal Recoverable Service Execution Solution Selection Algorithm, ORSESSA

Input: workflow WF, service execution solution ES_i (i = 1 … K).
Output: a service execution solution ES

```
1   generate SEG according to workflow WF and K
    service execution solutions;
2   S = 0; /* initial the largest weight sum S*/
3   ESSet = Ø;
4   for i = 1 to K do
5      calculate the edge weight sum of ESi, called Si;
6      if Si > S then
7         S = Si;
8         clear ESSet and put ESi into ESSet;
9      end if
```

```
10    if S_i == S then
11        add ES_i into ESSet;
12    end if
13  end for
14  if there are multiple solutions in ESSet then
15    choose a solution ES from ESSet, making
      ∑_{i=1}^{N} (Pos_i / N) * R_i largest; /* The more rearward
      position a service locates in CS, the lower success
      rate of recovery it will have after failed.*/
16    return ES;
17  end if
18  return the solution in ESSet;
```

Algorithm 1 searches from multiple service execution solutions for the one which has the largest weight sum. The larger the weight is, the more the backup paths will be, and the fewer the services which need compensation will be when switched from current solution to the backup path. Thus once invalid service occurs in this solution, the success rate of recovery will be relatively high.

In line 15, N denotes the number of ASs, Pos_i denotes the position of the node in solution ES, and R_i refers to the reliability of the node.

Assume there are E edges in each solution, the time complexity of Algorithm 1 is O(K*E).

C. Backup Path and Switch Cost

Normally multiple service execution solutions exist. We only choose one solution ES from them for actual execution and others are used to construct the backup paths. Once some WS fails in ES, ES could switch to another path at certain switch cost. Fig. 2 is an example SEG, in which S is the additional virtual source node and T is virtual destination node. In order to present backup path search algorithm, first we give the following definitions:

Definition 8. Backup Path Once failure occurs in current execution solution, the strategy will try to find a backup path according to the SEG. A backup path could be constructed by switching part of the current solution to another solution. So backup path also follows the service workflow and is composed of the edges in SEG.

Definition 9. Sequential Plan (SPlan) SPlan is a path from the source to the destination of a workflow, and it contains no more than one parallel branch. So usually a workflow includes multiple SPlans. For instance, Fig. 2 contains two sequential plans SPlan1 {S_1, S_2, S_3, S_4, S_7, S_8} and SPlan2 {S_1, S_2, S_5, S_6, S_7, S_8}.

Definition 10. Sequential Path (SPath) A sequential path could be formed by designating a concrete service for each abstract service in a sequential plan. It means a sequential path is a sequential sub path of the CCS.

In this paper, the designation is done by TSSA, so the sequential path is included in the service execution solution, e.g. service execution solution ES = (S_{11}; S_{21}; ((S_{31}; S_{41}) // (S_{51}; S_{61})); S_{71}; S_{81}) contains two sequential paths SPlan1 = (S_{11}; S_{21}; S_{31}; S_{41}; S_{71}; S_{81}), SPlan2 = (S_{11}; S_{21}; S_{51}; S_{61}; S_{71}; S_{81}).

Definition 11. Sequential paths switch Once certain WS fails in a sequential path, the CS could switch from one sequential path to another. Assume SPath1 = (S; … S_{ki}; …; T) and SPath2 = (S; … S_{kj}; …; T) are two sequential paths for the same sequential plan, where S_{ki} is the invalid node, and SPath2 does not involve S_{ki}. $S_{k'i}$ locates between S and S_{ki}, and it is the nearest node from S_{ki} among all the common nodes of SPath1 and SPath2. When SPath1 switches to SPath2, the services between $S_{k'i}$ and S_{ki} need to be compensated (excluding $S_{k'i}$ and S_{ki})

Definition 12. Illegal path switch The following three situations of path switch are regarded as illegal:

1) When a single service inside the parallel structure fails, the path switch could not involve the service outside the parallel structure in the same nesting level. It is to avoid the service in another branch of the parallel structure in the same nesting level being compensated. For instance, suppose S_{31} in SPath1 fails and S_{51} has been executed in SPath2 (Fig.2).
2) When a single service outside the parallel structure fails, the path switch could not involve the service inside the parallel structure in the same nesting level. It is to avoid the service in the parallel structure in the same nesting level being compensated. For instance, when S_{71} in service execution solution ES fails, both of the two sequential paths need to switch to other paths.
3) When multiple services inside the parallel structure fail, the path switching could not involve the service outside the parallel structure in the same nesting level yet.

Definition 13. Switch cost of service execution path The switch cost depends on both the compensation cost of compensated services and the workflow of the CS.

For example, we have service execution path P = (S_{11}; S_{21}; ((S_{31}; S_{41}) // (S_{51}; S_{61})); S_{71}; S_{81}). When S_{71} fails and the path is switched to Q = (S_{11}; S_{21}; ((S_{31}; S_{42}) // (S_{51}; S_{61})); S_{72}; S_{81}), S_{41} is compensated and the compensation cost is (ctime, ccost), which means it takes ctime and costs ccost to compensate S_{41}. Assume the QoS of path Q is (c, r, t), in which the QoS of parallel branch (S_3; S_4) and (S_5; S_6) is (c_1, r_1, t_1) and (c_2, r_2, t_2) respectively. Due to the compensation cost of S_{41}, the cost of path Q increases by ccost. The reliability of Q remains the same. The time needed to complete S_1, S_2, S_7 and S_8 (the rest services outside of the parallel structure) is [t-max(t_1,t_2)]. The parallel structure will take max(t_1+ctime,t_2) to complete after switching and adding the compensation cost. So the total time needed by path Q turns to [t+max(t_1+ctime,t_2)-max(t_1,t_2)]. Therefore, the QoS of Q after switching is (c+ccost, r, t+max(t_1+ctime,t_2)-max(t_1,t_2)).

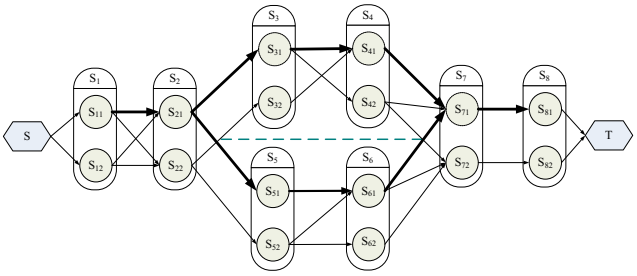

Figure 2. Example Service Execution Graph

The Backup Path Search Algorithm is presented below.

Algorithm 2. Backup Path Search Algorithm, BPSA

Input: composite service workflow WF, invalid nodes set V, current execution solution ES
Output: backup path B

```
1  generate sequential plan set SPlan_i(i = 1 … m) of WF; /*Assume there are m sequential plan in WF*/
2  generate sequential path set SPath_i(i = 1 … m) of ES;
3  B = ES /* initialize backup path B*/
4  for v_i ∈ V do
5     if v_i is outside the parallel structure then
6        get subSEG which is the subgraph of SEG between v_i and the parallel structure in the same nesting level with v_i;
7        find the first backup path B_i in subSEG; /*by searching precursor node from v_i continuously */
8        if no B_i is found then return NULL;
9     end if
10    if v_i is inside the parallel structure then
11       get subgraph of SEG called subSEG which is the parallel structure branch where v_i locates;
12       find the first backup path B_i in subSEG; /*by searching precursor node from v_i continuously */
13       if no B_i is found then return NULL;
14    end if
15 end for
16 substitute parts of the backup path B with B_i;
17 if conflict occurs then return NULL;
18 if B does not satisfy the QoS constraint then return NULL;
19 return B;
```

In BPSA, line 6 and line 11 restrict the search of backup path in a fixed area (sequential SEG). The edges connected with the invalid node in SEG are deleted, together with the unconnected path. Line 7 and line 12 continuously search the precursor node from v_i until the precursor node has a descendant node or beyond the range of subSEG. Thus the backup path obtained is the one with the least compensation cost. Line 18 checks the QoS of backup path B and guarantees the returned path satisfies the QoS constraint. The time complexity of the algorithm is O(|V|*N), where N is the number of ASs, V is the number of invalid WSs.

The backup path could guarantee the atomic consistency. Assume WS S_i fails in the current service execution solution ES_1, the algorithm finds the precursor node S_j and part of the path after S_j is switched to ES_2. Suppose the two parts of ES_1 and ES_2 are merged in S_n. S_k is a WS in front of S_j in solution ES_1. Because ES_1 is a solution of TSSA, it is corresponding to a transactional composite service. Thus the path from S_k to S_j satisfies the atomic consistency. On the other side, ES_2 is also a solution of TSSA, so the path from S_j to S_n in ES_2 satisfies the atomic consistency as well. Since S_j exists in both the two parts, the merged path from S_k to S_n satisfies the atomic consistency. Similarly, the path after S_n could be merged without breaking the atomic consistency. So the backup path after switching could guarantee the atomic consistency of transaction.

D. Local Induction

Local induction algorithm tries to progressively find a best service execution path P using positive feedback. The output P may satisfy the QoS constraint, or it is the best one in the QoS attributes. The algorithm also guarantees the transactional constraint of CS through construction and processing rules.

Algorithm 3. Local Induction Algorithm, LIA

Input: composite service workflow WF, invalid node set V, current execution solution ES
Output: service execution path P

```
1  topologically sort the ASs in WF; /* Line 10 will follow the order of the ASs and select the substitution service from the corresponding service set of AS*/
2  P = ES /*initialize the service execution path P*/
3  for v_i ∈ V do
4     call ASSFA to get the possible transaction property set of substitution service TPSet;
5     choose service s from the service set which has the shortest Euclidean distance from v_i, and satisfies TP(s) ∈ TPSet , update path P;
6  end for
7  if QoS(P) satisfies constraint then
8     update the SEG, return P;
9  else then /* this indicates the path P must have certain property of QoS that exceeds the limit */
10    while QoS(P) does not satisfy the constraint && there is service left in P that has not been substituted or executed do
11       update TPSet based on ASSFA;
12       utility(s) = w_1 * s.cost + w_2 * s.reliability + w_3 * s.time; /* w_1, w_2 and w_3 are dynamic weight*/
13       choose service s with the largest utility and satisfying TP(s) ∈ TPSet, update path P;
14    end while
15 end if
16 return path P and update the SEG;
```

QoS(P) denotes the QoS property of service execution path P and utility(s) denotes the utility of WS s.

The calculation of w_1, w_2 and w_3 refers to the universal approximation theorem with the sigmoid function [15]. It is described below.

E_1 = (QoS(P).cost - C)/C ,
E_2 = (R-QoS(P).reliability)/R ,
E_3 = (QoS(P).time - T)/T.
$F(u) = 1 / (1+e^{-u})$, let $W = F(E_1) + F(E_2) + F(E_3)$.
$w_1 = F(E_1) / W$, $w_2 = F(E_2) / W$, $w_3 = F(E_3) / W$.

The main idea of LLA is positive feedback. Its time complexity is O((|V|+N)*Q) where Q is the size of service set. The algorithm tries to select the path that may satisfy the constraint progressively through comparing the obtained QoS(P) and the constraint.

E. Recovery Algorithm

Algorithm 4. Service Recovery Based on Substitution Algorithm, SRBSA

Input: composite service workflow WF, invalid node

set V, current execution solution ES
Output: a path for recovery

```
1   B = BPSA(WF, V, ES); /*call Algorithm 2 to get the backup path B*/
2   if B == NULL then
3      P = LIA(WF, V, ES); /*call Algorithm 3 to get the execution path P if there is no backup path*/
4      return P;
5   end if
6   return B;
```

The service recovery algorithm calls Algorithm 2 first to get the backup path B (may be empty). If there is certain backup path found, it would be used as the final execution path. If not, call Algorithm 3 to get the execution path P. P could accomplish the recovery once it satisfies the QoS constraint.

V. EXPERIMENTAL ANALYSIS

In order to evaluate the performance of SRBSA, this section will compare SRBSA with Regin-Based Reconfiguration Algorithm (RBRA) introduced in [12] on the success rate of recovery. RBRA does the most similar work with this paper. Some comparisons have proved RBRA have certain advantages over the algorithms given in [10, 11] on the execution time and success rate of recovery. So we just compare SRBSA with RBRA. The experiment runs in the PC machine with Pentium(R) Dual 3GHz processor, 2G memory and Windows XP operating system under 100M LAN. The QoS of each WS contains cost, time and reliability. The time and reliability parameter of the QoS attribute is from the data which was released by Eyhab Al-Masi et al. [16-17]. The cost parameter generates at random within [1, 300]. The transactional property of each WS was generated among the set {p, c, r, cr} randomly. The experiment program is coded in Java. As RBRA did not consider the compensation cost, we do not take the compensation cost into account in this experiment.

In the following parts we will compare the algorithm success rate of recovery under sequential workflow (experiment 1) and complex workflow (experiment 2) respectively. As the time complexity of these two algorithms is both below 100ms and the error under Windows XP is relatively large (usually 15-16ms), we cannot have an accurate comparison on the execution time about the two algorithms. So the paper only compares the success rate of recovery.

A. Comparison under Sequential Pattern

Experiment 1 utilizes the sequential workflow containing 50 ASs as input, each of the ASs having the WS size of 50. First, get K service execution solutions for the workflow using TSSA. And further select one from K using ORSESSA. Then select an optimal execution path for RBRA using lp_solve [18], an open source integer programming framework. This path is equivalent to that obtained by service selection. The QoS constraints in both two algorithms are the same. In the sequential pattern, at most one service may fail at the same time. In order to make a more targeted evaluation on the performance of the algorithm, we do not randomly generate the position of invalid service, but designate it manually. This paper has designed 11 test cases, designating invalid service in position 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45 of the service execution solution ES respectively. Each of the cases runs for 500 times to calculate the success rate of recovery as formula (1). The result of experiment is shown as Fig. 3.

success rate of recovery = times of successful recovery / 500 (1)

It can be seen from Fig. 3 that the success rates of recovery in all the test cases are above 90%. As the position number of invalid service increases, the success rates of both SRBSA and RBRA decline. In general, SRBSA has a slightly higher success rate of recovery than RBRA. Although when the invalid service occurs at the tail of the CS, the success rate of SRBSA is not better than RBRA, this only begins to appear in test case 9 (position 37). It shows a general better success rate of recovery using SRBSA.

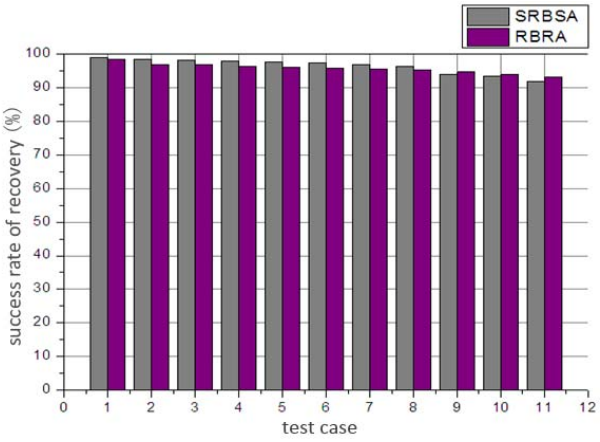

Figure 3. Success rate of recovery comparison under sequential pattern

B. Comparison under Complex Pattern

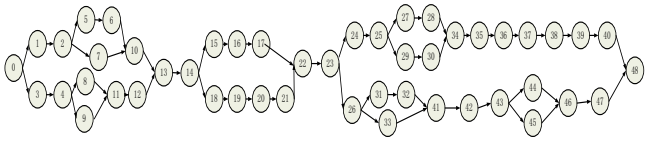
Figure 4. Complex workflow

Experiment 2 utilizes the complex workflow shown in Fig. 4 as input, containing 49 ASs. In the complex pattern, there may be more than one service failing at the same time. As the same as experiment 1, we have also designed 11 test cases and they are divided into 4 groups shown in table 1. At most 4 services may fail simultaneously in the workflow shown in Fig. 4. Experiment 2 evaluates the success rates of recovery respectively when the number of the invalid services is different. The result is present in Fig. 5. Again, the success rates of recovery in all the test cases are above 90%. As the number of invalid services increases, both SRBSA and RBRA have lower recovery success rate. The recovery success rates of SRBSA and RBRA are almost the same in each test case.

TABLE I. Test Case

Group No.	Invalid service count	Position numbers of invalid services	Case count	Case number
1	1	5, 16, 31	3	1-3
2	2	(16,20), (27, 44), (28,30)	3	4-6
3	3	(6, 7, 8), (24, 31, 33), (28, 29, 32)	3	7-9
4	4	(5, 7, 8, 9), (27, 30, 44, 45)	2	10-11

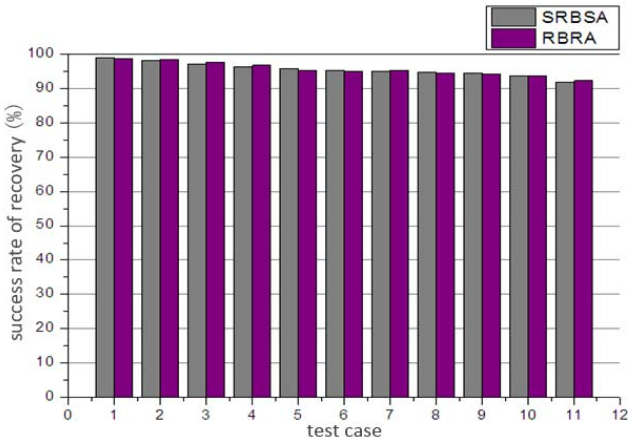

Figure 5. Success rate of recovery comparison under complex pattern

C. Conclusion of Experiment

In sequential pattern, the recovery success rate of SRBSA declines when the position number of invalid service increases. In parallel pattern, it declines not only as the position number of invalid service increases, but also as the number of invalid services rises. In general, the recovery success rates of SRBSA and RBRA are similar, and SRBSA performs better in sequential pattern. However, RBRA does not consider about the compensation cost of the service and it cannot guarantee the atomic consistency of the CS yet. SRBSA in this paper solves these problems.

VI. CONCLUSION

As the autonomy and heterogeneity of WS, the failure of service cannot be avoided when executing CS. This paper presents a service recovery algorithm based on substitution (SRBSA). On the basis of a global optimization service selection algorithm (TSSA), SRBSA takes construction and processing rules into account to guarantee atomic consistency of CS. This paper first presents the selection method of service execution solution. Then the concepts of the backup path and switch cost are introduced. At last, based on Backup Path Search Algorithm (BPSA) and Local Induction Algorithm (LIA), the service recovery algorithm SRBSA is described. Simulation experiment shows SRBSA in this paper is efficient and has high reliability. Moreover, it could guarantee the QoS constraint and the atomic consistency of transaction. In future work, we will consider the dependency relation in CS, not only between WSs but also between different QoS properties within the same WS. Based on the dependency relation, more practical strategy about transactional CS processing could be raised.

ACKNOWLEDGMENT

This work is supported by National Key Basic Research program of China under Grants No. 2010CB328104 and 2009CB320501, National Natural Science Foundation of China under Grants 61272531, 61070158, 61003257, 61060161, 61003311. China National Key Technology R&D Program under Grants No. 2010BAI88B03, China Specialized Research Fund for the Doctoral Program of Higher Education under Grants No. 20110092130002, China National S&T Major Project under Grants No. 2009ZX03004-004-04, State Key Laboratory of Information Security, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No.BM2003201, and Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No. 93K-9.

REFERENCES

[1] Alexandre Alves, Assaf Arkin, Sid Askary, et al. Web Services Business Process Execution Language (BPEL4WS) 2.0, 2007.

[2] Hossein Rahmani, Hassan Abolhassanni. Composite Web Service Failure Recovery Considering User Non-Functional Preferences [C]. Proceedings of 4th International Conference on Next Generation Web Service Practices. Seoul, Republic of Korea, 2008.

[3] Jiuxin Cao, Gongrui Zhu, Xiao Zheng, Bo Liu, Fang Dong. TASS: Transaction Assurance in Service Selection. 2012 IEEE 19th International Conference on Web Services(ICWS 2012), Honolulu,Hawaii,USA.24-29 June 2012.472-479

[4] T. Yu, K.J. Lin. Service selection algorithms for composing complex services with multiple QoS constraints [C]. Proceedings of ICSOC. Amsterdam, Holland, 2006.

[5] Deron Liang, Chen-Liang Fang, et al. Fault tolerant Web service [C]. Proceedings of the 10th Asia-Pacific Software Engineering Conference Software Engineering Conference. Chiang Mai, Thailand, 2003.

[6] X. Ye, Y. Shen. A middleware for replicated Web services [C]. Proceedings of IEEE International Conference on Web Services. Orlando, Florida, USA, 2005.

[7] Jorge Salas, Francisco Perez-Sorrosal, et al. WS-replication: a framework for highly available Web services [C]. Proceedings of the 15th International Conference on World Wide Web. Edinburgh, Scotland UK, 2006.

[8] Jiuxin Cao, Biao Zhang, Bo Mao, Bo Liu. Constraint Rules-based Recovery for Business Transaction [C]. Proceedings of GCC. Nanjing, China, 2010.

[9] G. Canfora, M. D. Penta, R. Esposito, M. L.a Villani. QoS-Aware Preplanning of Composite Web Services [C]. Proceedings of the IEEE International Conference on Web Services. Orlando, Florida, USA, 2005.

[10] T. Yu and K. Lin. Adaptive algorithms for finding replacement services in autonomic distributed business processes [C]. Proceedings of the 7th International Symposium on Autonomous Decentralized Systems. Chengdu, China, 2005.

[11] K.-J. Lin, J. Zhang, Y. Zhai. An Efficient Approach for Service Process Reconfiguration in SOA with End-to- End QoS Constraints [C]. Proceedings of the IEEE Conference on Commerce and Enterprise Computing. Vienna, Austria, 2009.

[12] Jing Li, Dianfu Ma, et al. Adaptive QoS-Aware Service Process Reconfiguration [C]. Proceedings of the IEEE International Conference on Services Computing. Washington DC, USA, 2011.

[13] S. Mehrotra, R. Rastogi, H. Korth, et al. A transaction model for multidatabase systems [C]. Proceedings of ICDCS. Yokohama, Japan, 1992.

[14] M. Dorigo, M. Birattari, T. Sttzle. Ant colony optimization, artificial ants as a computational intelligence technique [J] IEEE Computational Intelligence Magazine, 2006, 1(4): 28-39.

[15] G. Cybenko. Approximations by superpositions of sigmoidal functions. Mathematics of Control, Signals, and Systems, 2 (4): 303-314.

[16] E. Al-Masri, Q. H. Mahmoud. Discovering the best web service [C]. Proceedings of 16th International Conference on World Wide Web. Banff, Albert, CANADA, 2007.

[17] E. Al-Masri, Q. H. Mahmoud. QoS-based discovery and ranking of Web services [C]. Proceedings of IEEE 16th International Conference on Computer Communications and Networks. Honolulu, Hawaii, USA, 2007.

[18] K. E. Michel Berkelaar and P. Notebaert, Open source linear programming system, Sourceforge. http://lpsolve.sourceforge.net/.