International Journal of Innovative
Computing, Information and Control
Volume **8**, Number **7(B)**, July **2012**

ICIC International ©2012 ISSN 1349-4198
pp. **5265–5275**

FAULT-TOLERANT EXECUTION PLANNING FOR COLLABORATIVE BUSINESS PROCESSES BASED ON GENETIC ALGORITHMS

Jeyeon Oh[1], Nam Wook Cho[2,*], Hoontae Kim[3] and Suk-Ho Kang[4]

[1]Research Center
Hankook Delcam Ltd.
Guro 3-dong, Guro-gu, Seoul 152-768, South Korea
jy@delcam.co.kr

[2]Department of Industrial and Information Systems Engineering
Seoul National University of Science and Technology
172 Gongreung 2-dong, Nowon-gu, Seoul 139-743, South Korea
*Corresponding author: nwcho@seoultech.ac.kr

[3]Department of Industrial and Management Engineering
Daejin University
1007 Hoguk-ro, Pocheon-si, Gyounggi-do 487-711, South Korea
hoontae@daejin.ac.kr

[4]Department of Industrial Engineering
Seoul National University
1 Gwanak-ro, Gwanak-gu, Seoul 151-744, South Korea
shkang@snu.ac.kr

Received March 2011; revised July 2011

Abstract. *In the present study, we developed a method that provides, while minimizing costs, guaranteed-reliable execution plans for collaborative business processes conducted via web services. To that end, physical- and time-redundancy techniques are utilized and dynamic modifications of execution plan are provided. In order to address the dynamic execution planning problem, known to be NP-hard, we also developed a Genetic Algorithm (GA), the effectiveness of which was demonstrated through a set of experiments. Specifically, the GA was shown to be capable of providing near-optimal solutions in polynomial time. The main contribution of this paper is the more general execution planning method developed in the present study. While previous research assumed that the execution cost, time, and reliability of web services are the same, we relaxed that assumption. We expect that this will facilitate the application of our method in practice.*
Keywords: Collaborative business process, Quality of service, Web service, Fault-tolerance, Genetic algorithm

1. **Introduction.** A collaborative business process often is executed not only by internal processes but also via external web services [7,10-12]. Collaborating with a number of outside partners through web services requires sophisticated management of QoS (Quality of Service) aspects such as execution time, cost, reliability, availability, and others. Among the QoS aspects of a collaborative business process, reliable process execution has become more important [10]. For example, for a healthcare service process in which a number of partners collaborate through web services, reliable execution must be guaranteed. Although web service selection methods can be applied to web service QoS management, they do not guarantee reliable execution.

For the purposes of composite web service dependability [5], this paper presents a methodology that provides for dynamic execution plans at run-time. Previous studies

on web service dependability have sought to achieve fault-tolerant service provision by means of either exception handling through compensation [8,13,17] or physical redundancy [3,4,14,18]. Dependability was achieved to some extent, but only by adapting fault-tolerance to existing applications.

Laranjeiro [6] and Salatge [15] have dealt with fault-tolerance of composite web services. They relied mainly on web service replication, wherein a service provider can replicate the same web services from multiple servers. However, their fault-tolerant web composition method's application is limited, since they ignored cost and time constraints.

In our previous paper [10], we presented a dynamic execution planning approach that incorporates failure masking by redundancy for reliable execution of collaborative business processes. This approach, notwithstanding its contributions, has a limitation, in that it assumes that the execution cost, time, and reliability of web services are the same. Since it is more general and practical to assume that different web services have different characteristics, the execution planning problem needs to be addressed within that specific context. In this paper, we provide a methodology for reliable execution of business processes in a collaborative environment wherein an organization executes its processes through collaboration with outside partners via web services. The problem description and formulation are presented, and a Genetic Algorithm (GA) approach is developed to address the problem. Finally, a set of experiments is conducted to demonstrate the effectiveness of the proposed method.

2. Fault-Tolerance Model for Collaborative Processes.

In this section, a fault-tolerant execution planning problem is defined and its formulation is presented.

2.1. Problem description.

Let us suppose collaborative process execution of a sequential-flow business process composed of N activities, as illustrated in Figure 1. The reliability of the process is estimated to be $\prod r_i$ [1], where $0 \leq r_i \leq 1$ is the reliability of activity A_i. To complete the collaborative process, each activity can be executed by web services from inside or outside an organization [10].

In general, the management of a collaborative process is controlled by multiple workflow engines [2,9], which requires complicated and sophisticated control of process execution. For the simplicity of our process execution model, rather than dealing with multiple workflow engines, let us suppose that a business process is executed by a single organization in a collaborative environment in which its execution relies on collaboration with partners outside of the organization.

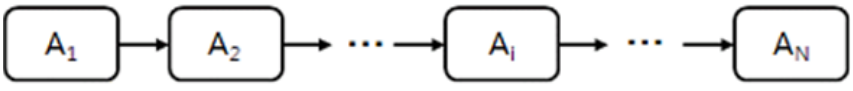

FIGURE 1. Sequential-flow business process

The reliability of an activity is often less than 1 and, therefore, reliable completion of the entire process is not always guaranteed. Thus, this study aims to develop a fault-tolerance model for collaborative processes that guarantees successful completion. Of course, a web service selection problem can be utilized to satisfy the reliability constraint of the process. The web service selection problem, however, neglects failures of web services; for example, if just one service fails, the entire process instance can fail [10]. Thus, web service selection often fails to provide for "reliable execution".

Before discussing our fault-tolerance model in detail, it is necessary to define the term "web service failure" as included in our model. Five types of web service failures can be identified [16]. They include *Crash Failure, Omission Failure, Timing Failure, Response*

Failure, and Byzantine Failure. Note that neither *Response Failure* nor *Byzantine Failure* is considered in our model.

In this paper, to provide more reliable execution of a collaborative process even in cases of web service failures, we propose a fault-tolerant model that dynamically modifies a web service execution plan. Two types of redundancy are considered in our model: time redundancy and physical redundancy. The time-redundancy method re-executes a web service in the case of a failure. Whereas it is a relatively simple method, its application to a collaborative process involving a due date can be limited, especially for a transactional process in which the failure of one service is correlated with other, subsequent activities [10]. By contrast, the physical-redundancy method utilizes extra services to render an activity fault-tolerant. Despite the effectiveness of this method, it incurs additional costs by requesting extra services. Therefore, in the present study, we employed combinations of time and physical redundancy to find the optimal execution plan for a process involving a due date.

Although the use of redundancy can greatly increase the reliability of a process, it is still insufficient to deal with the dynamics of web service execution. An original plan, no matter how optimized it has been, needs to be modified as a process is being executed. Therefore, in addition to redundancy, our proposed approach modifies an original execution plan at run-time. Figure 2 schematizes an original execution plan for a collaborative process composed of N activities. For example, for reliable execution of an activity A_1, the plan is supposed to request eight web services (ws_1, ws_8, ws_3, ws_{12}, ws_7, ws_{14}, ws_{21}, ws_{51}) by combining time and physical redundancies. Note that the eight web services are not requested simultaneously. Rather, by dividing the execution of services into three trials, the system offers reliable execution of A_1 at minimum cost [10]. However, if one of the services in the first or second trial completes successfully, the remaining plan needs to be adjusted at run-time. Suppose that ws_{12} completes successfully in the second trial of A_1, as illustrated in Figure 3, and that the third trial for A_1 in the original execution plan becomes unnecessary; thus, the execution of the services ws_{14}, ws_{21}, ws_{51} will be excluded from the new plan. Accordingly, the original plan should be re-optimized based on the current situation, shown in Figure 3.

The use of redundancy and dynamic modification of an execution plan are our main strategies for fault-tolerant execution planning of collaborative business processes. The challenge is to provide an optimal plan for reliable execution at minimal cost, which plan is to be modified during run-time.

2.2. **Formulation.** This section provides a detailed formulation of our approach.

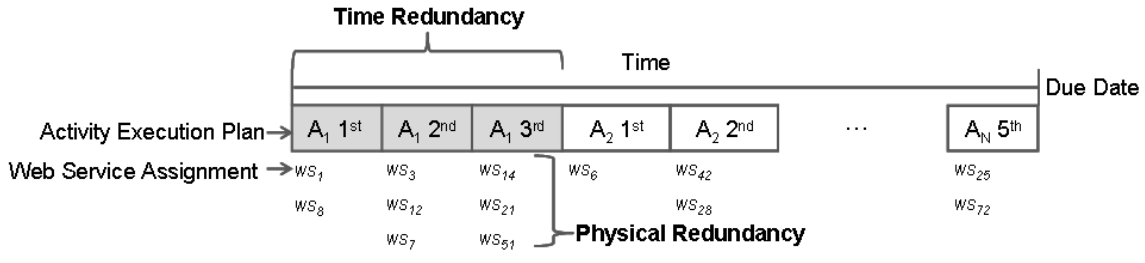

FIGURE 2. Execution planning with redundancies

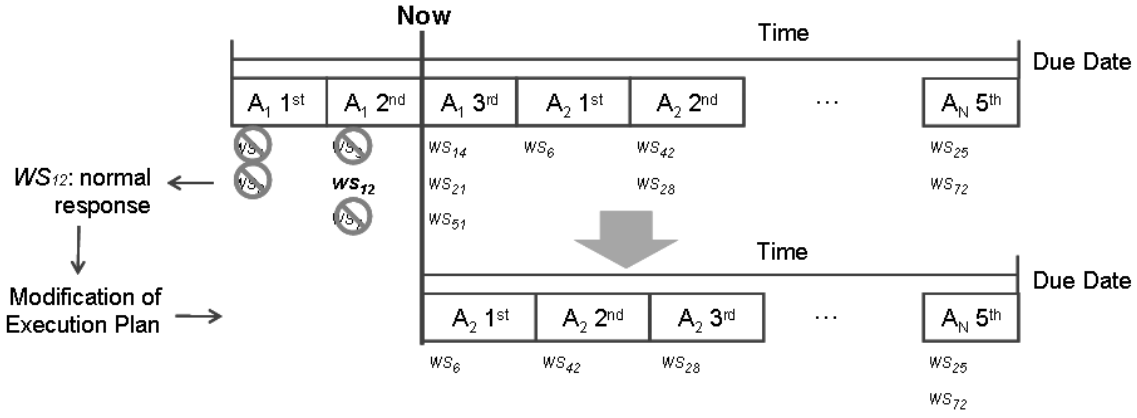

FIGURE 3. Modification of execution plan

Nomenclature

A_i	ith activity of a process
c_{ik}	execution cost of a web service k for A_i
t_{ik}	execution time of a web service k for A_i
λ_{ik}	probability of successful completion (reliability) of a web service k for A_i
x_i	decision variable that indicates the number of trials (time redundancy) for A_i
y_{ijk}	binary decision variable that indicates a web service k is executed on the jth trial for A_i
TD	time-to-deadline of the process
P	penalty rate charged per time beyond the deadline
R	reward rate awarded per saved time to the deadline

In our previous paper [10], it was assumed that the time, cost, and reliability of web services are the same. However, this assumption can be far from the reality, as different web services have different characteristics. Therefore, in our new model, each web service has its own time, cost, and reliability.

Let us assume that there are a sufficient number of vendors that provide web services for execution of an activity, and that an arbitrary web service k for activity A_i has its execution cost c_{ik}, time t_{ik}, and probability of successful completion λ_{ik}, where $0 \leq \lambda_{ik} < 1$. Let TD denote the time-to-deadline of the process, P the penalty rate charged per time beyond the deadline and R the reward rate awarded per saved time to the deadline. The total cost of a process execution plan is composed of two parts: execution cost and penalty/reward.

$$Total\ Cost = Execution\ Cost + (Penalty\ or\ Reward)$$

Execution cost is incurred by requests of web services; that is, requesting multiple web services would increase the execution cost but provide better reliability. In addition to the execution cost, it is assumed that a penalty is charged if a plan misses a deadline and, conversely, a reward is given if a plan completes its execution before a deadline. Penalty and reward were introduced to facilitate on-time delivery of business processes. Let EC_i denote the expected execution cost of activity i, and ET_i the expected execution time of

activity i. Then, the total cost of the process is represented as

$$E(Total\ Cost) = \sum_{i=1}^{N} EC_i + P \cdot \max\left\{0, \sum_{i=1}^{N} ET_i - TD\right\} + R \cdot \min\left\{0, \sum_{i=1}^{N} ET_i - TD\right\}. \tag{1}$$

Since our model assumes that different web services have different cost, time, and reliability, in addition to the number of requested web services for each activity, the execution plan also needs to determine which web services should be requested, from available vendors, for the activity. Therefore, the decision variables include x_i, which indicates the number of trials for A_i, and a binary variable y_{ijk} that indicates that a web service k is executed on the jth trial for A_i. Let p_{ij}^s and p_{ij}^o denote the probability of success and occurrence of the jth trial for A_i, respectively. Then, p_{ij}^s and p_{ij}^o are calculated as

$$p_{ij}^o = \prod_{l=1}^{j-1} \prod_{k} (1 - \lambda_{ik} y_{ilk}) \text{ and } p_{ij}^s = p_{ij}^o \left(1 - \prod_{k} (1 - \lambda_{ik} y_{ijk})\right). \tag{2}$$

Therefore, the expected cost (EC_i) and time (ET_i) for A_i can be obtained as follows:

$$EC_i = \sum_{j=1}^{x_i} p_{ij}^o \left(\sum_{k} c_{ik} y_{ijk}\right). \tag{3}$$

$$ET_i = \sum_{j=1}^{x_i} \left[p_{ij}^s \sum_{l=1}^{j} \max_k\{t_{ik} y_{ilk}\}\right]. \tag{4}$$

By plugging Equations (3) and (4) into (1), the execution planning problem is formulated as follows:

Minimize:

$$\sum_{i=1}^{N} \sum_{j=1}^{x_i} p_{ij}^o \left(\sum_{k} c_{ik} y_{ijk}\right) + P \cdot \max\left\{0, \sum_{i=1}^{N} \sum_{j=1}^{x_i} \left[p_{ij}^s \sum_{l=1}^{j} \max_k\{t_{ik} y_{ilk}\}\right] - TD\right\}$$
$$+R \cdot \min\left\{0, \sum_{i=1}^{N} \sum_{j=1}^{x_i} \left[p_{ij}^s \sum_{l=1}^{j} \max_k\{t_{ik} y_{ilk}\}\right] - TD\right\} \tag{5}$$

Subject to:

$$\prod_{j=1}^{x_i} \prod_{k} (1 - \lambda_{ik} y_{ijk}) \leq \varepsilon, \text{ for every } i.$$

$$y_{ijk} = 0 \text{ or } 1, \text{ for every } i, j, \text{ and } k.$$

Note that to avoid generating an infinite number of services for an activity, we additionally assume that an activity does not fail if its probability of successful completion exceeds a threshold, $(1 - \varepsilon)$, where ε, $0 < \varepsilon < 1$, is an arbitrarily small number.

Oh et al. [10] showed that a well-known Knapsack problem can be transformed into a special case of a simple execution planning problem where the cost, time, and reliability of web services are the same and it can be deduced that the planning problem is NP-hard. And, as our problem presented in Equation (5) can be reduced to the simple execution planning problem, it also can be deduced that our problem is NP-hard. In the next section, a Genetic Algorithm (GA) is provided to address the execution planning problem.

3. **Genetic Algorithm (GA) for the Execution Planning Problem.** To allow the GA to search for a solution to our problem, we first need to encode the problem with a suitable genome. In our case, the genome is represented by a matrix set for which the number of matrices is equal to the number of activities. The matrix in our GA represents an execution plan for the corresponding activity. In each matrix, a row represents a trial for an activity, and a column indicates a web service provider. $Cell(i, j) = 1$ if a service for the ith trial is requested of the jth service provider; $Cell(i, j) = 0$, otherwise. Figure 4 illustrates the genome encoding scheme.

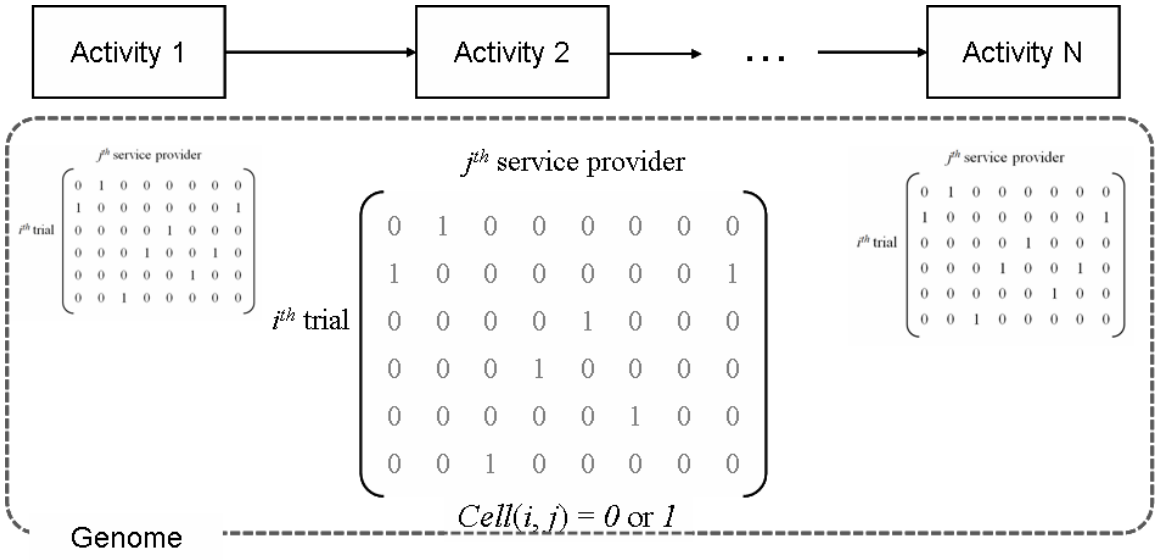

FIGURE 4. Genome encoding

The generation procedure for an initial population is as follows:

- The number of service requests for each trial has to be at least one; the number of web service requests for each activity has to be at most one. For each row (i.e., trial), a column (i.e., web service) is randomly selected from unselected columns, and then the cell value is set to 1. Note that the sum of a row is at least one, which means that at least one service should be selected from "available" web services for each trial.
- Select a column for which its sum is 0, randomly select cells in the column, and set the value to 1 until the success rate of an activity is greater than the threshold, $(1-\varepsilon)$. Note that the success rate of an activity has to be greater than the threshold.

The objective function of Equation (5), defined in Section 2, is used as a fitness function. This function represents the expected total cost of a process execution plan, which cost is composed of two parts: execution cost and penalty/reward.

In the proposed GA, the binary tournament selection mechanism is applied as a selection operator. In this scheme, two individuals are randomly chosen, and that the one which represents the better solution is selected as the first parent. To obtain the second parent, the procedure is repeated.

In our GA, two crossover operators are used. The first crossover operator, called Strategy-Interchange Crossover (SIC), interchanges two matrixes (i.e., two execution plans) for the selected activity, as shown in Figure 5. As a result of the crossover, for example, Child 1 has a different genome from Parent 1 beyond the crossover point.

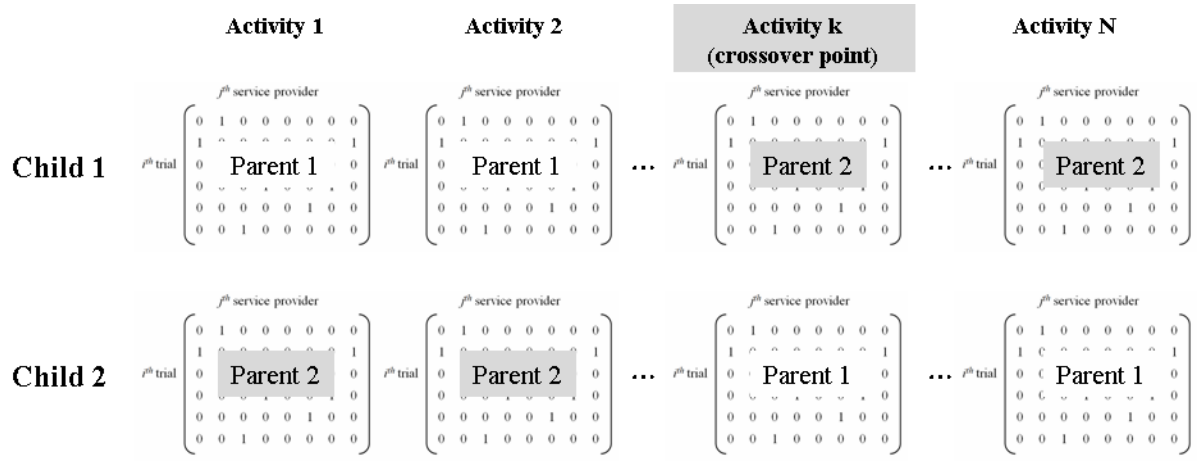

FIGURE 5. Strategy-interchange crossover (SIC)

The second crossover operator, called Strategy-Mix Crossover (SMC), mixes two matrixes (i.e., two execution plans) for the selected activity, as shown in Figure 6. With this operator, half of the rows (i.e., trials) should be repaired. The repair process is as follows:

- If the sum of a row is 0, randomly select a cell and modify the cell value to 1.
- If the sum of a column is more than 2, modify the cell values so that the sum of the column is 1.
- If the sum of a column is 0, select randomly a cell and then modify the cell value to 1 until the success rate of an activity is above the threshold, $(1 - \varepsilon)$.

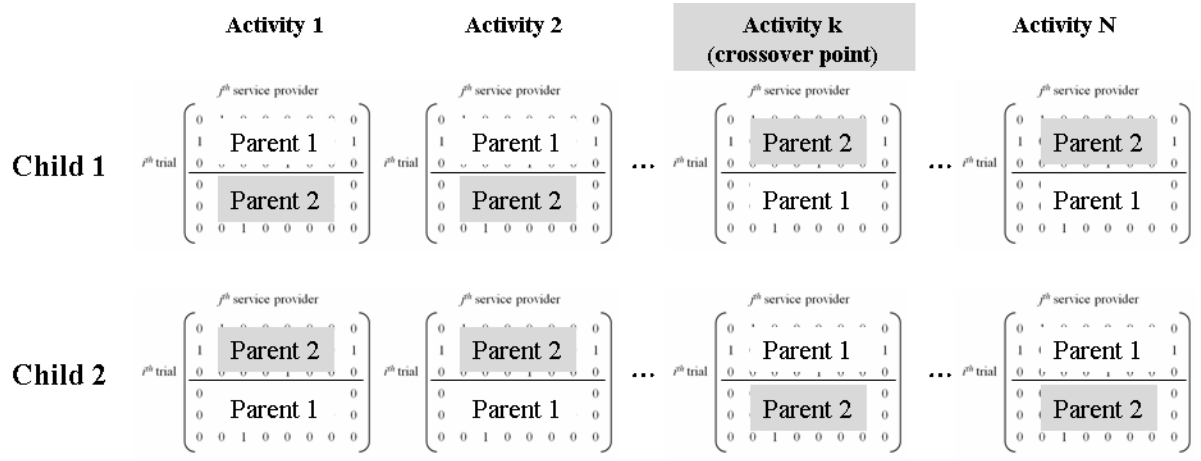

FIGURE 6. Strategy-mix crossover (SMC)

While the crossover operators combine the strategies of two parent genomes, the mutation operator randomly selects an activity (i.e., a matrix in the genome) and randomly replaces an execution plan. The application of GA is explained in the next section.

4. **Experiments.** In this section, we analyze the performance of the proposed GA under various conditions such as deadline, penalty/reward, and web service variability. Also, the crossover operators were compared. Table 1 summarizes the experimental settings employed.

Our GA (implemented by JAVA) was set for a population of 200 individuals, a crossover probability of 0.5 and a mutation probability of 0.05. The GA stops when an objective

TABLE 1. Experimental settings

Parameter	Setting	Acronym	Description
Deadline	Loose	Loose-D	The deadline is two times greater than the sum of the average execution times of all of the activities.
	Tight	Tight-D	The deadline is equal to the sum of the average execution times of all of the activities.
Penalty or Reward	High	High-PR	The penalty/reward is 10 times greater than the expected execution cost for a service.
	Low	Low-PR	The penalty/reward is equal to the expected execution cost for a service.
Web Service Variability (Execution Time)	High	High-TV	The coefficient of variation of the web service execution times is equal to 0.5.
	Low	Low-TV	The coefficient of variation of the web service execution times is equal to 0.3.
Web Service Variability (Reliability)	High	High-RV	The failure probability of a web service k for an activity i, $1-\lambda_{ik}$, is uniformly distributed in $[\varepsilon^{\wedge(1/6)}, \varepsilon^{\wedge(1/3)})$, where $\varepsilon = 0.0001$. To guarantee reliable execution of an activity i, a minimum of 4 and a maximum of 6 web services are required.
	Low	Low-RV	The failure probability of a web service k for an activity i, $1-\lambda_{ik}$, is uniformly distributed in $[\varepsilon^{\wedge(1/4)}, \varepsilon^{\wedge(1/3)})$, where $\varepsilon = 0.0001$. To guarantee reliable execution of an activity i, at least 4 web services are required.

value has no change over 2,000 iterations. We used Windows XP on an Intel Core2 Duo 2.66 GHz CPU with 2 GB RAM. The algorithm was run 40 times on each problem setting. For the experiment, the number of activities (N) and the number of available web services for each activity (M) were set at 20 and 30, respectively.

Table 2 compares the performances of the two crossover schemes, SIC and SMC. To show the difference, the ratios of fitness values, (*fitness using SIC*)-(*fitness using SMC*)/(*fitness using SMC*), are observed with iterations. On average, the GA terminated about 10,000 iterations with SIC and about 8,000 with SMC. As can be seen, SMC showed a better performance overall, but the difference narrowed to less than 0.1 percent with more than 8,000 iterations.

The experimental results show that there is very little difference between the crossover operations under their termination conditions. However, if dynamic modifications of execution plans are required, the running time of the GA becomes important. Figure 7 plots the results of comparison experiments for 200 individuals and 2,000 iterations.

As can be seen in Figure 7, SMC exhibits better performance when the variability of web service reliability is high. Higher variability increases the number of feasible execution plans and, consequently, increases the size of a problem. We believe that this could cause the performance difference between SMC and SIC.

Finally, we validated the effectiveness of the proposed GA by comparing it with an optimal solution obtained by an exhaustive search method. The numbers of activities (N) and available web services for each activity (M) were set at 4 and 6, respectively. The experiment was conducted under the tight deadline (Tight-D) and high penalty/reward

TABLE 2. Performance comparisons of crossover schemes (%)

Settings				Iterations			
Deadline	Penalty/ Reward	Time Variability	Reliability Variability	1,000	2,000	4,000	8,000
Tight	High	Low	Low	1.304	0.571	0.186	−0.003
Loose	High	Low	Low	0.395	0.143	0.061	−0.058
Tight	Low	Low	Low	0.306	−0.070	−0.291	−0.110
Loose	Low	Low	Low	0.169	0.148	0.034	−0.081
Tight	High	High	Low	0.298	0.101	−0.026	0.073
Loose	High	High	Low	0.127	0.064	0.007	0.022
Tight	Low	High	Low	0.147	0.073	0.061	0.105
Loose	Low	High	Low	−0.043	0.067	0.023	0.015
Tight	High	Low	High	2.087	1.031	0.347	0.266
Loose	High	Low	High	0.666	0.323	0.165	0.036
Tight	Low	Low	High	1.968	0.959	1.092	0.654
Loose	Low	Low	High	0.547	0.319	0.053	−0.065

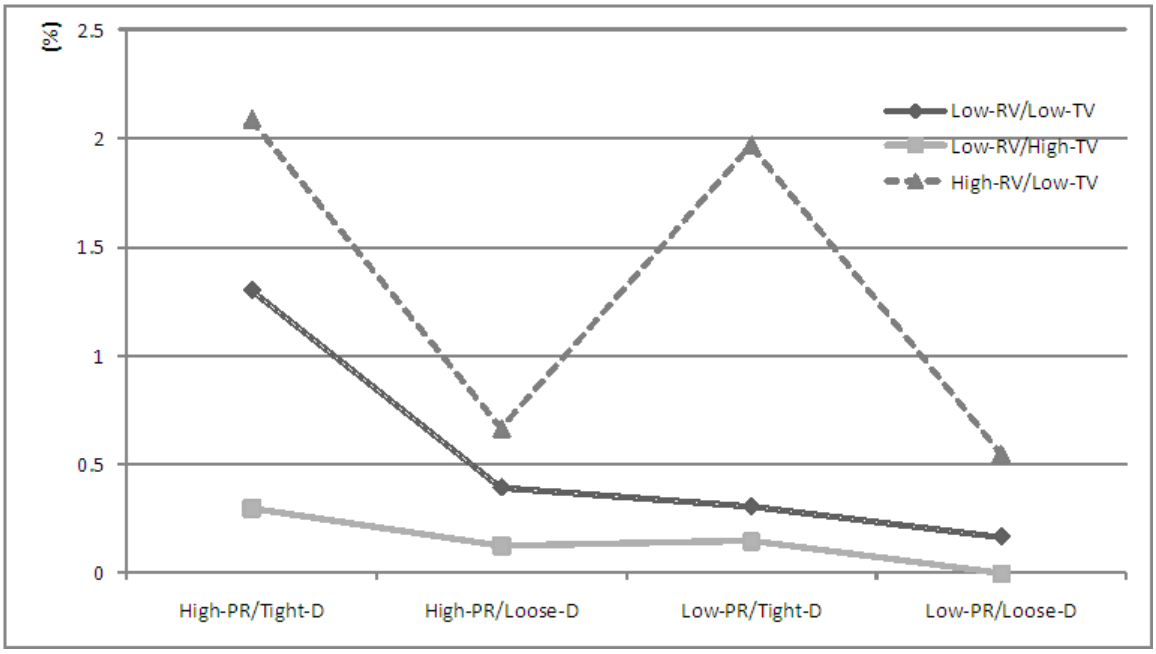

FIGURE 7. Performance comparisons with 2,000 iterations

(High-PR) conditions. The results showed that the proposed GA reached the optimal solution within 4,000 iterations. Note that in this experiment, the problem size was reduced in order to obtain an optimal solution through total enumeration.

4.1. **Application example.** In this section, to illustrate the implementation of our GA on a business process, an application example is provided. Let us suppose a business process composed of five activities, whose execution relies on collaborating with twenty web services. Each web service has different execution cost, execution time, and reliability[1]. The deadline of the business process is 80; both the penalty rate charged per time beyond the deadline (P) and the reward rate awarded per saved time to the deadline (R) are 500.

[1]The application example data can be provided upon request to the corresponding author.

If reliability is a priority, a web service selection problem would choose a web service with the highest reliability for each activity. With the application of the web service selection method, the execution of the business process would result in the following: reliability = 0.557, execution cost = 7361.5 and execution time = 99. However, the application of our GA, shown in Table 3, can guarantee the reliable execution within the deadline. For the given example, the reliability of the execution plan is $1 - \varepsilon$, where ε is an arbitrarily small number ($\varepsilon = 0.0001$). The expected execution cost and time is 13,443 and 67, respectively. It should be noted that the increase in the execution cost can be compensated by higher reliability and faster completion prior to deadline.

Table 3. Execution plan generated by GA

Activity	Trial	Selected Services	Activity	Trial	Selected Services
Activity 1	1st	10, 18	Activity 4	1st	7, 10
	2nd	16		2nd	17
	3rd	14		3rd	4, 14
Activity 2	1st	7, 9	Activity 5	1st	7
	2nd	10		2nd	17
	3rd	5, 12		3rd	16
Activity 3	1st	6, 20		4th	4
	2nd	10		5th	10
	3rd	4	–	–	–
	4th	1	–	–	–

5. **Conclusions.** In this paper, we developed a method that utilizes physical and time redundancies to provide reliable execution plans for collaborative business processes. Our method not only provides a reliable execution plan but also can dynamically modify an original plan at run-time. To address the dynamic execution planning problem, known to be NP-hard, a Genetic Algorithm (GA) was developed, and its effectiveness was demonstrated through a set of experiments.

The main contribution of this paper is the development of a more general execution planning method, which combines time redundancy with physical redundancy in order to increase the reliability of execution plan of collaborative process by employing GA. Whereas the previous research assumed that the execution cost, time, and reliability of web services are the same, we relaxed that assumption. This, we expect, will facilitate the application of our method in practice. Also, for the GA, two different proposed crossover schemes were compared in a set of experiments. The effectiveness of the proposed GA was validated also by comparison with an optimal solution.

We must admit, however, that our idea has some limitations. Generally, a collaborative business process is controlled by multiple organizations and multiple workflow engines, but our model assumes a special case: a single organization controls a collaborative business process the execution of which entails collaboration with outside partners. Reliable execution of a collaborative business process controlled by multiple organizations and requiring complicated and sophisticated control of process execution, therefore, would be a suitable topic for future research.

Another limitation is the fact that our model deals mainly with web service failures to provide reliable business process execution. The reliability of a collaborative business process depends on not only web services but also software artifacts responsible for process execution, for example, a workflow engine. It should be noted the failures of such artifacts can lead to failure of the entire process.

A third limitation is our model's neglect of the execution cost incurred in guaranteeing reliable execution of collaborative business processes. This neglect, not surprisingly, can result in excessive execution costs. Thus, a more comprehensive framework that considers a greater variety of QoS aspects would be called for.

There is still another research issue to be dealt with. In our model, we assumed that there exist a sufficient number of available web services to request. However, in reality, this is not always guaranteed. The relaxation of that assumption, then, would extend the applicability of the proposed method.

REFERENCES

[1] J. Cardoso, A. Sheth, J. Miller, J. Arnold and K. Kochut, Quality of service for workflows and web service processes, *Web Semantics: Science, Services and Agents on the World Wide Web*, vol.1, no.3, pp.281-308, 2004.

[2] Q. Chen and M. Hsu, Inter-enterprise collaborative business process management, *Proc. of International Conference on Data Engineering*, pp.253-260, 2001.

[3] D. Jayasinghe, *FAWS for SOAP-Based Web Services: A Client-Transparent Fault Tolerance System for SOAP-Based Web Services*, http://www.ibm.com/developerworks/webservices/library/ws-faws/, 2005.

[4] L. Juszczyk, J. Lazowski and S. Dustdar, Web service discovery, replication, and synchronization in ad-hoc networks, *Proc. of the 1st International Conference on Availability, Reliability and Security*, pp.847-854, 2006.

[5] H. Kopetz and P. Verissimo, Real time and dependability concepts, in *Distributed Systems*, 2nd Edition, Workingham, Addison-Wesley, 1993.

[6] N. Laranjeiro and M. Vieira, Towards fault tolerance in web services compositions, *Proc. of the Workshop on Engineering Fault Tolerant Systems, Article No. 2*, 2007.

[7] L. Lei and Z. Duan, Automating web service composition for collaborative business processes, *Proc. of the 11th International Conference on Computer Supported Cooperative Work in Design*, pp.894-899, 2007.

[8] L. Lin and F. Liu, Compensation with dependency in web services composition, *Proc. of NWeSP*, pp.183-188, 2005.

[9] C. Liu, Q. Li and X. Zhao, Challenges and opportunities in collaborative business process management: Overview of recent advances and introduction to the special issue, *Information Systems Frontier*, vol.11, pp.201-209, 2009.

[10] J. Oh, N. W. Cho, H. Kim, Y. Min and S. H. Kang, Dynamic execution planning for reliable collaborative business processes, *Information Sciences*, vol.181, no.2, pp.351-361, 2011.

[11] J. Oh, J. Jung, N. W. Cho, H. Kim and S. H. Kang, An integrated process modeling for dynamic B2B collaboration, *Knowledge-Based Intelligent Information and Engineering Systems, LNAI*, vol.3683, pp.602-608, 2005.

[12] C. Peltz, Web services orchestration and choreography, *Computer*, vol.36, no.10, pp.46-52, 2003.

[13] P. F. Pires, M. R. F. Benevides and M. Mattoso, Building reliable web services compositions, in *Web, Web-Services, and Database Systems 2002, LNCS*, A. Chaudhri, M. Jeckle and E. Rahm (eds.), vol.2593, 2003.

[14] J. Salas, F. P. Sorrosal, M. P. Martinez and R. J. Peris, WS-replication: A framework for highly available web services, *Proc. of the 15th International Conference on World Wide Web*, pp.357-366, 2006.

[15] N. Salatge and J. C. Fabre, Fault tolerance connectors for unreliable web services, *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp.51-60, 2007.

[16] A. S. Tanenbaum and M. van Steen, Fault tolerance, in *Distributed Systems: Principle and Paradigms*, NJ, Prentice Hall, 2002.

[17] V. Issarny, F. Tartanoglu, A. Romanovsky and N. Levy, Coordinated forward error recovery for composite web services, *Proc. of SRDS*, pp.167-176, 2003.

[18] X. Ye, Providing reliable web services through active replication, *Proc. of ICIS*, pp.1111-1116, 2007.