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
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 $\prod_r^1$, 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](image)

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 \( (w_{s1}, w_{s8}, w_{s3}, w_{s12}, w_{s7}, w_{s14}, w_{s21}, w_{s51}) \) 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 \( w_{s12} \) 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 \( w_{s14}, w_{s21}, w_{s51} \) 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](image-url)
Figure 3. Modification of execution plan

**Nomenclature**

- $A_i$: $i$th 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 $A_i$
- $\lambda_{ik}$: 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 $j$th 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 $\lambda_{ik}$, 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

\[
E(TotCost) = \sum_{i=1}^{N} EC_i + P \cdot \max \left\{ 0, \sum_{i=1}^{N} ET_i - TD \right\}
\]

\[
+ R \cdot \min \left\{ 0, \sum_{i=1}^{N} ET_i - TD \right\}.
\] (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 \( j \)th trial for \( A_i \). Let \( p^s_{ij} \) and \( p^o_{ij} \) denote the probability of success and occurrence of the \( j \)th trial for \( A_i \), respectively. Then, \( p^s_{ij} \) and \( p^o_{ij} \) are calculated as

\[
p^o_{ij} = \prod_{l=1}^{j-1} \prod_{k} \left(1 - \lambda_{ik} y_{ljk} \right) \quad \text{and} \quad p^s_{ij} = p^o_{ij} \left(1 - \prod_{k} \left(1 - \lambda_{ik} y_{ljk} \right) \right).
\] (2)

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

\[
EC_i = \sum_{j=1}^{x_i} p^o_{ij} \left( \sum_{k} c_{ik} y_{ijk} \right).
\] (3)

\[
ET_i = \sum_{j=1}^{x_i} \left[ p^o_{ij} \sum_{l=1}^{j} \max_{k} \{ t_{ik} y_{ljk} \} \right].
\] (4)

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

Minimize:

\[
\sum_{i=1}^{N} \sum_{j=1}^{x_i} p^o_{ij} \left( \sum_{k} c_{ik} y_{ijk} \right) + P \cdot \max \left\{ 0, \sum_{i=1}^{N} \sum_{j=1}^{x_i} \left[ p^o_{ij} \sum_{l=1}^{j} \max_{k} \{ t_{ik} y_{ljk} \} \right] - TD \right\}
\]

\[
+ R \cdot \min \left\{ 0, \sum_{i=1}^{N} \sum_{j=1}^{x_i} \left[ p^o_{ij} \sum_{l=1}^{j} \max_{k} \{ t_{ik} y_{ljk} \} \right] - TD \right\}
\] (5)

Subject to:

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

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

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 \( \varepsilon, 0 < \varepsilon < 1 \), 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 \(i\)th trial is requested of the \(j\)th service provider; \(Cell(i, j) = 0\), otherwise. Figure 4 illustrates the genome encoding scheme.

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, \((1 - \epsilon)\). 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.
The second crossover operator, called Strategy-Mix Crossover (SMC), mixes two matrices (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, $(1 - \varepsilon)$.

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

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

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, $(\text{fitness using SIC}) - (\text{fitness using SMC}) / (\text{fitness using SMC})$, 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 (%)

<table>
<thead>
<tr>
<th>Settings</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,000</td>
</tr>
<tr>
<td>Deadline</td>
<td>Penalty/Reward</td>
</tr>
<tr>
<td>Tight</td>
<td>High</td>
</tr>
<tr>
<td>Loose</td>
<td>High</td>
</tr>
<tr>
<td>Tight</td>
<td>Low</td>
</tr>
<tr>
<td>Loose</td>
<td>Low</td>
</tr>
<tr>
<td>Tight</td>
<td>High</td>
</tr>
<tr>
<td>Loose</td>
<td>High</td>
</tr>
<tr>
<td>Tight</td>
<td>Low</td>
</tr>
<tr>
<td>Loose</td>
<td>Low</td>
</tr>
<tr>
<td>Tight</td>
<td>High</td>
</tr>
<tr>
<td>Loose</td>
<td>High</td>
</tr>
<tr>
<td>Tight</td>
<td>Low</td>
</tr>
<tr>
<td>Loose</td>
<td>Low</td>
</tr>
</tbody>
</table>

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 – ε, where ε is an arbitrarily small number (ε = 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

<table>
<thead>
<tr>
<th>Activity</th>
<th>Trial</th>
<th>Selected Services</th>
<th>Activity</th>
<th>Trial</th>
<th>Selected Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity 1</td>
<td>1st</td>
<td>10, 18</td>
<td>Activity 4</td>
<td>1st</td>
<td>7, 10</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>16</td>
<td></td>
<td>2nd</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>3rd</td>
<td>14</td>
<td></td>
<td>3rd</td>
<td>4, 14</td>
</tr>
<tr>
<td>Activity 2</td>
<td>1st</td>
<td>7, 9</td>
<td></td>
<td>1st</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>10</td>
<td></td>
<td>2nd</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>3rd</td>
<td>5, 12</td>
<td></td>
<td>3rd</td>
<td>16</td>
</tr>
<tr>
<td>Activity 3</td>
<td>1st</td>
<td>6, 20</td>
<td></td>
<td>4th</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>10</td>
<td></td>
<td>5th</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3rd</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4th</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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