2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

Dependency-based Risk Evaluation for Robust Workflow Scheduling

Mingzhong Wang*, Kotagiri Ramamohanarao† and Jinjun Chen‡
*Beijing Lab of Intelligence Information Technology
School of Computer Science
Beijing Institute of Technology
Beijing 10081, China
Email: wangmz@bit.edu.cn
†Department of Computer Science and Software Engineering
The University of Melbourne
Victoria 3010, Australia
Email: rao@csse.unimelb.edu.au
‡Faculty of Engineering and Information Technology
University of Technology, Sydney
PO Box 123, Broadway NSW 2007, Australia
Email: Jinjun.Chen@uts.edu.au

Abstract

The robustness of a schedule, with respect to its probability of successful execution, becomes an indispensable requirement in open and dynamic service-oriented environment, such as grids or clouds. We design a fine-grained risk assessment model customized for workflows to precisely compute the cost of failure of a schedule. In comparison with current course-grained model, ours takes the relation of task dependency into consideration and assigns higher impact factor to tasks at the end. Thereafter, we design the utility function with the model and apply a genetic algorithm to find the optimized schedule, thereby maximizing the robustness of the schedule while minimizing the possible risk of failure. Experiments and analysis show that the application of customized risk assessment model into scheduling can generally improve the successful probability of a schedule while reducing its exposure to the risk.

1. Introduction

Workflow technology has become the primary unifying mechanism to support modeling, management, and execution of both organizational business processes and large-scale scientific computing, such as climate modeling, astrophysics, medical surgery and disaster recovery. Meantime, as the rise of service-oriented computing and cloud computing for their cost-efficiency and flexibility, tasks in workflow definitions are usually needed to be delegated to various services out of the organizational boundary for execution, thus shifting the execution environment from being static and trustworthy to open, dynamic, and non-deterministic. Consequently, flooded with large numbers of services with different capabilities, costs, reputations and reliability, workflow schedulers should not only find a solution which satisfies deadline or budget constraints from users, but also need to guarantee the schedule to be carried out successfully at the maximum potential.

Some research work proposed to use the concept of trustworthiness or reputation to model and measure the probability that each service will fulfill its commitments and promises from its historical behaviors, and incorporate trustworthiness into the scheduling process to improve the reliability and robustness of the schedule [1]–[3]. Currently, there are mainly two strategies to guarantee the reliability of a schedule. One is to set a reputation threshold to select eligible services only. Another is to treat all task nodes in the workflow independently, and then multiply their probability of success as the indicator of schedule reliability.

However, these approaches have not taken the workflow structure into consideration, thus losing the op-

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.286

portunity to find better solutions. For example, a task at the end of a workflow should get more attention or priority than a task at the beginning, since the failure of the ending one will ruin all previous effort. Correspondingly, the scheduler should allocate more resources (time or money) for the ending task to assure its successful completion. Therefore, more fine-grained methods are required to support precise measurement for the risk of service failure.

Addressing this issue, this paper introduces a risk evaluation model according to workflow structure by assigning different impact factors or weights to tasks based on their ordering relationship. The model is rooted in risk assessment and risk management in social and economic study, but customized for the workflow context. Thereafter, a robust scheduling process, which uses the model to estimate the cost of unreliable services in the schedule, is proposed to improve the reliability and robustness of the schedule.

The remainder of this paper is organized as follows. Section 2 provides an overview of the state of the art. Section 3 presents and discusses details on the risk evaluation model, as well as its application in scheduling process for improved robustness and reliability. The design of experiment and results are demonstrated and discussed in Section 4. Finally, in Section 5 we present our conclusions and perspective for future work.

2. Related work

Various heuristic methods of workflow scheduling have been extensively studied to find an optimized resource allocation for performance improvement [4]. To work in non-deterministic open environment, some propose to apply trust to improve the schedule robustness and reliability, thus reducing the probability of rescheduling [1]–[3]. However, their use of trust, or probability that a service will succeed, is coarse-grained that the relation of task dependency is not considered. In comparison, we provide a fine-grained risk assessment model customized for workflows by assigning higher weight to tasks at the end.

Risk management is a common business practice, and should be part of QoS (Quality of Service) management. However, the common strategy in QoS for risk management is rescheduling or retrying alternative services [5], [6], neglecting the precise calculation of cost of failure for each decision. Therefore, our model can be integrated into existing workflow management systems for QoS assurance.

3. Risk evaluation for robust workflow scheduling

In this section, the concept and algorithm of risk assessment model for workflows are explained. The section first provides the preliminary knowledge about workflows, and then presents the methods to recursively compute the cost of failure for a workflow schedule. Thereafter, a simple case is studied to provide an empirical insight. Finally, a genetic algorithm incorporating the model is designed and applied.

3.1. Workflow model

A workflow process can be specified as a directed acyclic graph (DAG). In case of a workflow with iterations, its corresponding cyclic graph can be converted into a DAG by loop unrolling.

Definition 1 (Workflow graph definition): A workflow is abstracted as a DAG, which is a quadruple $G =< V, E, t_0, t_n >$ consisting of following elements:

- V is a finite set of vertices. Each vertex is associate with an executable atomic task $t_i, 0 \leq i \leq n$. To simplify the discussion and without loss of generality, we assume that tasks associated with different vertices are different.
- $E \subset V \times V$ is a set of edges connecting vertices in the graph. An edge, which is a triple $e =< t_i, c, t_j > \in E$, defines a precedent relation between two tasks. It means task t_j will start if and only if t_i completes successfully and the condition c is satisfied.
- $t_0 \in V$ is the starting node of the graph.
- $t_n \in V$ is the ending node, signifying the completion or termination of the workflow.

It is required that each workflow has only one entry point, and completes at a single end node. With the condition c, the edge formalism in the definition can support all three types of workflow constructs, including sequence, branch, and parallel. The DAG definition for a workflow in fact defines a partial order among tasks.

Definition 2 (Precedence relation): The edge $e =< t_i, c, t_j > \in E$ in a DAG defines a precedence relation between tasks, denoted as $t_i \xrightarrow{c} t_j$. Task t_j is the successor of t_i, and t_i is the predecessor of t_j. If the condition c is always true, the notation is abbreviated as $t_i \rightarrow t_j$.

For the purpose of this paper, we assume all precedence relations are unconditional to simplify the discussion. All conclusion and results can be extended to include conditional cases.

Definition 3 (Execution path): In a DAG, a set of tasks with the relation of $(t_i \rightarrow t_{i+1}) \wedge (t_{i+1} \rightarrow t_{i+2}) \wedge ... \wedge (t_{j-1} \rightarrow t_j)$ forms an execution path $\pi_{i...j} = (t_i, t_{i+1}, t_{i+2}, ..., t_j)$.

3.2. Risk evaluation model

Risk assessment, which evaluates quantitative or qualitative value of risk related to a concrete situation and a recognized threat, is a common practice in business and social management. To extend and customize its usage in robust and reliable workflow scheduling, we first introduce the concept of the cost of failure.

Definition 4 (Cost of failure): The risk for performing a certain activity or decision is evaluated as the cost of failure, which is the multiplication of the potential loss L, and the probability p that the loss will occur. Formally, $cost_f = Lp$.

To simplify the discussion, we assume in this paper that the potential loss is the execution cost of each task, ignoring the possible chaining impact outside the workflow. Therefore, L for each task equals to its running cost.

For a given execution path, failure of any task on the path will cause the whole sequence to fail. Therefore, we introduce Theorem 1 to compute the cost of failure for the sequence, which helps developers to evaluate the risk and consequences of task failure.

Lemma 1: For a single task t which has the probability p to be completed for the price of $cost$, the cost of its failure is $cost_f^t = (1-p)cost$.

Proof: t has the probability of $1-p$ to fail. Therefore, its failure will statistically result in the loss of $(1-p)cost$. □

Lemma 2: In execution path $\pi_{1...n} = (t_1, t_2, \ldots, t_n)$, if each $t_i (1 \leq i \leq n)$ has the probability $p(i)$ to be completed for the price of $cost(i)$, then the failure of the k^{th} task will cost

$$cost_f^{t_k} = \prod_{i=1}^{k-1} p(i) \big[\sum_{j=1}^{k} cost(j)(1-p(k)) \big]$$

Proof: The commencement of t_k relies on the success of all previous tasks. Therefore, t_k has the probability of $\prod_{i=1}^{k-1} p(i)$ to be executed.

When t_k is executed, the total cost involved is $\sum_{j=1}^{k} cost(j)$, which has the probability of $1-p(k)$ to fail. Therefore, the cost of failure for t_k is $\prod_{i=1}^{k-1} p(i) \big[\sum_{j=1}^{k} cost(j)(1-p(k)) \big]$. □

Theorem 1: In execution path $\pi_{1...n} = (t_1, t_2, \ldots, t_n)$, if each $t_i (1 \leq i \leq n)$ has the probability $p(i)$ to be completed for the price of $cost(i)$, then the cost of failure for the path is

$$cost_f^{\pi_{1...n}} = \sum_{k=1}^{n} \Big(\prod_{i=1}^{k-1} p(i) \big[\sum_{j=1}^{k} cost(j)(1-p(k)) \big] \Big)$$

Proof: When $n = 1$, according to Lemma 1, the cost of failure should be $cost_f^{\pi_1} = cost_f^{t_1} = (1-p(1))cost(1)$ which is the same as the result from the theorem.

Let us assume $cost_f^{\pi_{1...(n-1)}}$, which represents the cost of failure for the execution path of first $n-1$ tasks, is correct. When t_n is added into the path, its failure will result in the cost of $cost_f^{t_n}$, which can be computed by Lemma 2. Therefore, the cost of failure for the path $\pi_{1...n}$ is the summation of $cost_f^{\pi_{1...(n-1)}}$ and $cost_f^{t_n}$.

$$\begin{aligned} cost_f^{\pi_{1...n}} =& cost_f^{\pi_{1...(n-1)}} + cost_f^{t_n} \\ =& \sum_{k=1}^{n-1} \Big(\prod_{i=1}^{k-1} p(i) \big[\sum_{j=1}^{k} cost(j)(1-p(k)) \big] \Big) \\ &+ \prod_{i=1}^{n-1} p(i) \big[\sum_{j=1}^{n} cost(j)(1-p(n)) \big] \\ =& \sum_{k=1}^{n} \Big(\prod_{i=1}^{k-1} p(i) \big[\sum_{j=1}^{k} cost(j)(1-p(k)) \big] \Big) \end{aligned}$$

Since both the basis and the inductive step have been proved, it has now been proved by mathematical induction that $cost_f^{\pi_{1...n}}$ holds for the execution path $\pi_{1...n}$. □

Theorem 1 provides the method to compute the cost of failure for sequential execution of tasks. Supplementally, Theorem 2 provides the way to compute the cost of failure for concurrent task execution.

Theorem 2: Let $t_1, t_2, \ldots, t_k$ be a group of tasks running concurrently, satisfying $(t_0 \rightarrow t_1) \wedge (t_0 \rightarrow t_2) \wedge ... \wedge (t_0 \rightarrow t_k)$ and $(t_1 \rightarrow t_n) \wedge (t_2 \rightarrow t_n) \wedge ... \wedge (t_k \rightarrow n)$. If each task $t_i (1 \leq i \leq k)$ has the probability $p(i)$ to be completed for the price of $cost(i)$, then the cost of failure for their concurrent execution is

$$cost_f^{\|_{t_1,\ldots,t_k}} = \sum_{i=1}^{k} cost(i) \big(1 - \prod_{i=1}^{k} p(i)\big)$$

Proof: When tasks are executed in parallel, the failure of any task will cause the whole system to fail. Therefore, the probability that the system succeeds is $\prod_{i=1}^{k} p(i)$. Meanwhile, the probability for the system to fail is $1 - \prod_{i=1}^{k} p(i)$.

During the execution, the involved cost is the summation of all tasks. Therefore, the cost of failure for parallel execution is $\sum_{i=1}^{k} cost(i) \big(1 - \prod_{i=1}^{k} p(i)\big)$. □

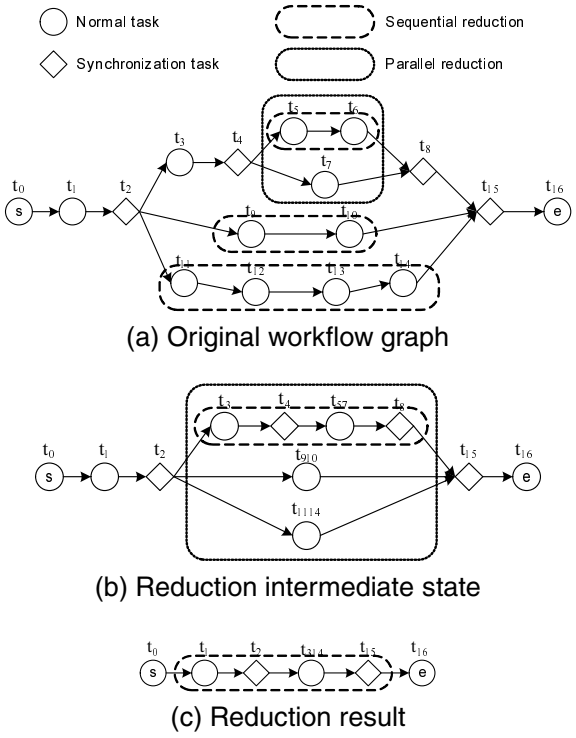


Figure 1. Recursively computing the cost of failure

Applying the concept of graph reduction and sub-workflow substitution [7], the cost of failure for the workflow can be computed recursively with Theorem 1 and Theorem 2. To simplify the discussion, we assume workflow graphs are well-structured [8] in the sense that it consists of matching pairs of a node that splits the flow and a node that joins the flow. [8] proves with corresponding transforming algorithms that all workflow graphs can be converted to be well-structured.

In a well-structured workflow, tasks can be categorized as either non-functional synchronization tasks or normal tasks. A synchronization task performs no real execution, but is used to control the split and merge of flow sequence. Therefore, synchronization tasks have the execution cost 0, and always complete successfully. Synchronization tasks generally have more than one predecessors or successors. In contrast, a normal task is used to complete the real business activity, and has only one predecessor or successor.

The process of workflow graph reduction is illustrated with an example in Figure 1. In the example, synchronization tasks are represented as diamonds and normal tasks as circles.

There are two operations for workflow graph reduction and risk computation, and the impact of synchronization tasks can be ignored during the process since they are not functional.

- Sequential reduction, which is enclosed by dashed boxes, uses a single task to substitute sequential tasks on a sub-flow or sub-path containing no synchronization task with multiple in- or out-edges. After the reduction of a path $\pi_{i...j}$, for the substitute task t_{ij}, its probability to complete successfully is $\prod_{k=i}^{j} p(k)$, execution cost is $\Sigma_{k=i}^{j} cost(k)$, and the cost of failure is $cost_f^{\pi_{i...j}}$. For example, the execution path $\pi_{9,10}$ is replaced by t_{910}, and $\pi_{11...14}$ is replaced by t_{1114} from Figure 1a to Figure 1b.
- Parallel reduction, which is enclosed by dotted boxes, uses a single task to substitute several parallel running paths, each of which contains only one normal task. After the reduction of a set of parallel tasks $t_i, \ldots, t_j$, for the substitute task t_{ij}, its probability to complete successfully is $\prod_{k=i}^{j} p(k)$, execution cost is $\Sigma_{k=i}^{j} cost(k)$, and the cost of failure is $cost_f^{\|_{t_i,\ldots,t_j}}$. For example, t_{56}, which is a substitution of execution path $\pi_{5,6}$, and t_7 are reduced to t_{57} from Figure 1a to Figure 1b, while t_{38}, which is a substitution of path $\pi_{3...8} = (t_3, t_4, t_{57}, t_8)$, and t_{910} and t_{1114} are reduced to t_{314} from Figure 1b to Figure 1c.

According to the definition of cost of failure and Theorem 1, it can be concluded that sequential reduction can be conducted incrementally with any grouping policy, and has the property of associativity. That is, for a sequence $t_1, t_2, \ldots, t_n$, the reduction can be carried out in several arbitrary sub-sequences, and then recursively performing sequential reductions. For example, in Figure 1a, the reduction of $\pi_{11...14} = t_{11}, t_{12}, t_{13}, t_{14}$ can be performed with the following steps $(t_{11}, ((t_{12}, t_{13}), t_{14}))$, in which parentheses stand for the order of reduction grouping.

However, parallel reduction does not have this property, thus all parallel tasks between the same pair of synchronization tasks must be substituted at once.

After applying sequential reduction and parallel reduction recursively, the workflow graph can finally be reduced to a task sequence (Figure 1c), on which the cost of failure for the workflow can be obtained.

Algorithm in Figure 2 shows the detailed computing process. $Pred(t_j) = \{t_i | t_i \rightarrow t_j\}$ is used to represent the set of predecessors of task t_j, while $Succ(t_i) = \{t_j | t_i \rightarrow t_j\}$ for the set of successors of t_i. Functions for task substitution, $SUBST\_PATH()$ and $SUBST\_PARALLEL()$, are defined in Figure 3.

3.3. Case analysis for risk evaluation model

The risk evaluation model proposed in this paper takes the task dependency into consideration. In com-

function WF_COST_FAILURE(G)
  repeat
    for all $t_i \in NormalTasks$ **do**
      ▷ Incrementally substitute sequential tasks
      if $t_j = Succ(t_i) \in SyncTasks$ **then**
        continue
      else
        t'_i = SUBST_PATH(t_i, t_j)
        replace t_i, t_j with t'_i in $NormalTasks$
      end if
    end for
    for all $t_i \in SyncTasks$ **do**
      $substitute = true$
      ▷ Remove redundant synchronization tasks
      if $|Pred(t_i)| == |Succ(t_i)| == 1$ **then**
        remove t_i from $SyncTasks$
        continue;
      end if
      ▷ Only perform substitution when all branches after a synchronization task are ready
      for all $t_j \in Succ(t_i)$ **do**
        if $(t_k = Succ(t_j) \notin SyncTasks) \vee t_k$ changed in loop **then**
          $substitute = false$
          continue;
        end if
      end for
      ▷ Perform the parallel substitution
      if $substitute = true$ **then**
        t'_i = SUBST_PARALLEL($Succ(t_i)$)
        replace $Succ(t_i)$ with t'_i in $NormalTasks$
      end if
    end for
  until ($|SyncTasks|$ is 0) $\wedge$ ($|NormalTasks|$ is 1)
end function

Figure 2. Algorithm for computing cost of failure for workflow

function SUBST_PATH($t_1, t_2, \ldots, t_n$)
  create new task t
  $Pred(t) = Pred(t_1)$, $Succ(t) = Succ(t_n)$
  $cost_f^t = cost_f^{\pi_{1\cdots n}}$ with Theorem 1
  $p(t) = \prod_{i=1}^{n} p(i)$, $cost(t) = \Sigma_{i=1}^{n} cost(i)$
  return t
end function

function SUBST_PARALLEL($t_1, t_2, \ldots, t_n$)
  create new task t
  $Pred(t) = Pred(t_1)$, $Succ(t) = Succ(t_1)$
  $cost_f^t = cost_f^{\|_{t_1,\ldots,t_n}}$ with Theorem 2
  $p(t) = \prod_{i=1}^{n} p(i)$, $cost(t) = \Sigma_{i=1}^{n} cost(i)$
  return t
end function

Figure 3. Functions for task substitution

parison, current work treats all tasks independently. To demonstrate the difference, we provide a case study to illustrate their impact on the scheduling results.

For a workflow consisting of n tasks, each of which has probability of $p(i)$ to succeed, any failure will cause the whole workflow to terminate exceptionally. Therefore, the probability that a schedule will succeed is $\prod_{i=1}^{n} p(i)$. We call this naive production.

Current work simply applies naive production to model the reliability of a schedule, neglecting the impact of task ordering. For example, for a sequence of tasks, the first task has the same impact as the last on the workflow since their cost of failure is not considered. However, in daily life, it is commonsense that the fail of last one will result in much worse situation. Applying the risk assessment, the cost of failure for the naive model should be $\Sigma_{i=1}^{n} cost(i)(1 - \prod_{i=1}^{n} p(i))$. It is the same as the formula in Theorem 2, which is designed for parallel tasks.

Addressing the disadvantage of current work, the risk evaluation in this paper applies a more precise model to differentiate the task occurrence. The major difference comes from Theorem 1 which assign higher weight on tasks occurring late.

Assuming a simplified workflow which containing 5 sequential tasks $t_1, t_2, \ldots, t_5$. To make the discussion simpler, let each task share the same probability p to succeed, and the same cost c for execution. Therefore, 3 functions can be obtained:

- The probability for the workflow to succeed: p^5,
- The risk of failure in naive production model: $5c(1 - p^5)$, and
- The risk of failure in our model: $c(1 + p + p^2 + p^3 + p^4 - 5p^5)$. The function is inferred directly from Theorem 1.

These functions are drawn in Figure 4 with $p \in [0, 1]$. In the proposed risk evaluation model, since the dependency or ordering relationship between tasks are considered, the final schedule will follow the rule that if a task is likely to fail, let it fail early. Therefore, there is a bell-like concave curve. In case that the successful probability for tasks at the beginning is not very low, but that for tasks at the end is not high enough, the cost of failure is in fact increasing steadily since more tasks will be executed for some cost while the overall

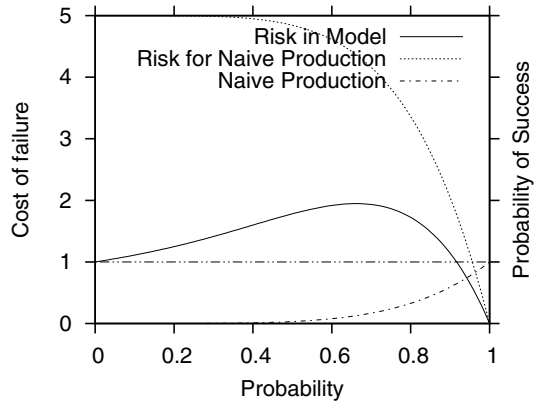


Figure 4. Case comparison for risk models

workflow will fail. After a certain stage in which the successful probabilities are high, the cost of failure falls dramatically since the whole workflow is likely to succeed.

In contrast with the bell-like concave curve for the cost of failure, the probability of success for the workflow is increasing steadily, making them uncorrelated. This may not be straightforward, but it resembles our life experience better which assigns higher priority to tasks at the end.

The figure also shows that with the same setting, the possible loss in our model is always lower than naive production model, especially in unreliable environment in which participants are likely to fail. As the system becomes more reliable, the gap between two models narrows down. Therefore, for the same probability of success for the workflow, our model always provides better and more precise risk assessment than naive model because of the consideration of dependency information of workflows.

Although the analysis is carried out in the artificial setting, the conclusions can still serve as an empirical insight into different models. Moreover, as the system becomes more complex with various task costs and successful probabilities, the risk model proposed in this paper would be more applicable since the dependency between tasks has greater impact on schedules.

3.4. Risk-aware workflow scheduling

In service-oriented environment, there are generally several services with different processing capacities eligible to complete each task in the workflow graph G. The selection of service s_i to execute task t_i results in a triplet $< time(i), cost(i), p(i) >$, representing its execution time, cost, and probability to succeed. Moreover, users may set the constraints of budget B and deadline D as part of their business logic in the workflow definition.

With the evaluation of the cost of failure, a risk-aware and robust scheduler can be defined. In essence, it is a multi-objective optimization problem addressing the following criteria.

$$Cost(G) = \sum_{t_i \in G} cost(i)$$
$$Time(G) = \sum_{T_i \in G_{CP}} time(i)$$
$$P(Success(G)) = \prod_{t_i \in G} p(i)$$
$$cost_f^G = WF\_COST\_FAILURE(G)$$

Maximize $P(Success(G))$ and minimize $cost_f^G$ subject to:

$$Cost(G) < B$$
$$Time(G) < D$$

$Cost(G)$ is the total cost of the overall schedule, and must be lower than the budget. $Time(G)$ is the total running duration of the schedule, and must be within the deadline. The computation of $Time(G)$ needs to detect the critical path of the workflow. CP is used to denote the critical path which goes along t_0 to t_n with the longest overall duration, and G_{CP} is the set containing all tasks on the critical path.

$P(Success(G))$ indicates the probability that the solution will finally succeed. It is a product of $p(i)$, which indicates the probability that each task will succeed, because the failure of any task will cause the whole schedule to fail. We assume that each service's probability of success is independent.

$cost_f^G$ is the cost of failure for the workflow, which is computed with the algorithm in Figure 2. The value represents the risk of a schedule with respect to the possible loss. With its application, the scheduler will assign higher weight to the tasks at the end since their failure will ruin all previous efforts.

Therefore, a cost-effective and robust scheduler should try to maximize $P(Success(G))$ and minimize $cost_f^G$. To find an optimized schedule for this multi-objective scheduling problem, we adopt genetic algorithm (GA) to design and implement the scheduling process.

Typically, a GA contains two main problem-dependent components: the encoding schema and the evaluation function. The encoding scheme breaks a potential solution into discrete parts that can vary independently. In GA, the discrete parts are called "genes" and the solution is called a chromosome. For

scheduling purpose, we treat each task in the workflow as a gene, and the vector of scheduled tasks as a chromosome.

The evaluation function, denoted as Ψ, measures the quality of a particular solution according to the given optimization objectives. Ψ contains two parts: the fitness function, denoted as $F(G)$, to encourage the higher robustness of the schedule, and the penalty function, denoted as $P(G)$, to enforce the constraints of budget B and deadline D. In our definition, the higher the value of Ψ, the better the solution.

$$
\begin{aligned}
F_{cost}(G) &= 1 - Cost(G)/B \\
F_{time}(G) &= 1 - Time(G)/D \\
F_{risk}(G) &= cost_f^G/Cost(G) \\
F(G) &= (1 - F_{risk}(G)) \cdot P(Success(G)) \\
&\quad \cdot (F_{time}(G) + F_{cost}(G)) \\
P_{budget}(G) &= \begin{cases} F_{cost}(G) & \text{if } Cost(G) >= B \\ 0 & otherwise \end{cases} \\
P_{deadline}(G) &= \begin{cases} F_{time}(G) & \text{if } Time(G) >= D \\ 0 & otherwise \end{cases} \\
P(G) &= P_{budget}(G) + P_{deadline}(G) \\
\Psi &= F(G) + P(G)
\end{aligned}
$$

In $F(G)$, the multiplication of $P(Success(G))$ and $F_{cost}(G) + F_{time}(G)$ indicates that the saving on execution time or cost is only valuable if the schedule succeeds. The result is further multiplied with $(1 - F_{risk}(G))$ to ensure the portion of saving on execution time or cost is in the safety zone. $P(G)$ is designed to enforce the bias against ineligible schedules which exceed the budget or go beyond the deadline. After the chromosome and the evaluation function are defined, standard GA operators can be applied to find the optimized schedule.

4. Experiments and evaluations

In this section, experiments were designed to evaluate reliable scheduling strategies with and without precise risk assessment. The environment for the experiments was created by adopting the DAG dataset of the Resource-Constrained Project Scheduling Problem [9].

In simulation, each task was supported with a set of different services with various cost, time and success probability levels (mean values). The cost level of each task was randomly selected between 100 and 2000, the time level was between 100 and 5000, and the probability level was set to 1.0. The exact value of cost, time and probability for each service were obtained by randomly fluctuating about 20% around the cost, time and trust level of its corresponding task. To make the simulated environment more practical, the fluctuation followed the observation that in commerce the execution time of a service is typically inversely proportional to its cost, while its success probability is in direct proportion to the cost.

To simulate different levels of user constraints, the budget B and the deadline D were evaluated at a wide range of possible values between the maximum and minimum allowed time and cost for the workflow. k_1 was used for the constraint levels of B, and k_2 for that of D. Both of them ranged from 0 to 1, the higher the value, the tighter the constraints.

$$
\begin{aligned}
B &= maxCost - k_1(maxCost - minCost) \\
D &= maxTime - k_2(maxTime - minTime)
\end{aligned}
$$

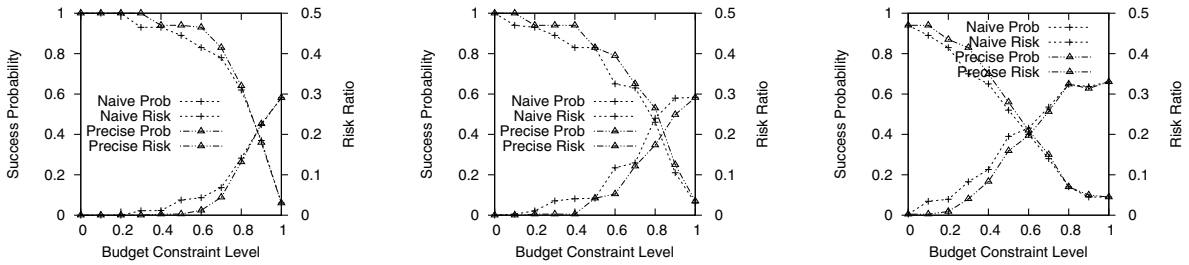
In the experiments, $maxCost$ and $minCost$ were obtained by always selecting the service with the most or least processing cost for each task respectively, while $maxTime$ and $minTime$ were obtained by selecting the slowest/fastest service for each task.

The implementation of the GA was based on [10]. The population for each generation was 50 and the total generation was 2000 for each experiment. All other parameters, such as crossover and mutation probability, were the system default.

The result was evaluated with two parameters. One was $P(Success(G))$ for the probability to succeed, and the other was the risk ratio $cost_f^G/Cost(G)$ for the percentage of cost of failure in the total cost. $cost_f^G$ for both scheduling strategies, with and without precise risk assessment, are computed with Algorithm in Figure 2. The higher, the better for $P(Success(G))$, but the lower, the better for risk ratio.

The experiments were carried out by setting k_2 at 0.2, 0.5 and 0.8 to represent the relaxed, medium and tight constraint of D respectively, and then increasing k_1 from 0 to 1 by 0.1. Figure 5 shows the results about the correlation between B, D, $P(Success(G))$, and risk ratio of schedules, in a randomly chosen DAG with 32 nodes. Altering the experiments by changing resource settings and DAG definitions yields similar results to Fig. 5.

The results show that $P(Success(G))$ and risk ratio get worse as the constraint levels become tighter for both types of schedulers. However, scheduling with precise risk evaluation performs better in general than naive robust scheduler, in terms that at the same constraints level, the precise model may find a better solution with higher $P(Success(G))$ and lower risk ratio. The improvement relies on the consideration of ordering relation between tasks, which makes bad schedules fail early. The gap between two models goes down as the constraint level tightens.



(a) Deadline Level = 0.2 (b) Deadline Level = 0.5 (c) Deadline Level = 0.8

Figure 5. Cost of failure at variant constraint levels

5. Conclusion and future Work

In this paper, we introduce a customized risk assessment model for workflows. In comparison with existing coarse-grained approaches targeting at improving schedule reliability and robustness, our model provides more precise risk evaluation by taking the task dependency into consideration and assigning higher weights to tasks at the end. Thereafter, we design the utility function with the model and apply a genetic algorithm to find the optimized schedule, thereby maximizing the robustness of the schedule while minimizing the possible cost of failure.

The experiments show that application of customized risk assessment model into scheduling can generally improve system robustness and reliability in comparison with current coarse-grained approach, especially when the constraint level is not very tight, meaning that the resulting schedule is more likely to succeed with less exposure to risk.

For future work, we intend to explore wider applications of the customized risk assessment model to improve QoS of the workflow management systems, especially in case of exceptions. Moreover, we plan to extend our model to embrace the impact of market-oriented resource management and SLA (Service Level Agreement) on risk management.

Acknowledgment

The research work reported in this paper is supported by National Science Foundation of China under Grant No. 61100172.

References

[1] M. Rahman, R. Ranjan, and R. Buyya, "Reputation-based dependable scheduling of workflow applications in peer-to-peer grids," *Comput. Netw.*, vol. 54, pp. 3341–3359, December 2010.

[2] M. Wang, K. Ramamohanarao, and J. Chen, "Trust-based robust scheduling and runtime adaptation of scientific workflow," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 16, pp. 1982–1998, 2009.

[3] X. Wang, C. S. Yeo, R. Buyya, and J. Su, "Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1124–1134, October 2011.

[4] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," in *Metaheuristics for Scheduling in Distributed Computing Environments*, F. Xhafa and A. Abraham, Eds. Springer Berlin Heidelberg, 2008, vol. 146, pp. 173–214.

[5] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528 – 540, 2009.

[6] L. Qi, W. Lin, W. Dou, J. Jiang, and J. Chen, "A QoS-aware exception handling method in scientific workflow execution," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 16, pp. 1951–1968, 2011.

[7] 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.

[8] J. Vanhatalo, H. Völzer, F. Leymann, and S. Moser, "Automatic workflow graph refactoring and completion," in *Proceedings of the 6th International Conference on Service-Oriented Computing*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 100–115.

[9] Project Scheduling Problem Library - PSPLIB, accessed on 28-January-2012. [Online]. Available: http://129.187.106.231/psplib/

[10] K. Meffert, N. Rotstan, C. Knowles, and U. B. Sangiorgi, JGAP - Java Genetic Algorithms and Genetic Programming Package, accessed on 28-January-2012. [Online]. Available: http://jgap.sourceforge.net/