Optimizing Makespan and Reliability for Workflow Applications with Reputation and Look-ahead Genetic Algorithm

Xiaofeng Wang[a,*], Chee Shin Yeo[b], Rajkumar Buyya[c], Jinshu Su[a]

[a] *School of Computer, National University of Defense Technology, China.*
[b] *Distributed Computing Group, Computing Science Department, Institute of High Performance Computing, Singapore.*
[c] *Cloud Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.*

Abstract

For applications in large-scale distributed systems, it is becoming increasingly important to provide reliable scheduling by evaluating the reliability of resources. However, most existing reputation models used for reliability evaluation ignore the critical influence of task runtime. In addition, most previous work uses list heuristics to optimize the makespan and reliability of workflow applications instead of Genetic Algorithms (GAs), which can give several satisfying solutions for choice. Hence, in this paper, we first propose the Reliability-Driven (RD) reputation, which is time-dependent and can be used to effectively evaluate the reliability of a resource in widely distributed systems. We then propose Look-Ahead Genetic Algorithm (LAGA) which utilizes the RD reputation to optimize both makespan and reliability of a workflow application. LAGA uses a novel evolution and evaluation mechanism: (i) the evolution operators evolve the task-resource mapping of a scheduling solution and (ii) the evaluation step determines the task order of solutions by using our proposed max-min strategy, which is the first two-phase strategy that can work with GAs. Our experiments show that the RD reputation improves the reliability of an application with more accurate reputations, while LAGA provides better solutions than existing list heuristics and evolves to better solutions more quickly than a traditional GA.

Keywords: reliability, reputation, workflow scheduling, genetic algorithm, heuristic

1. Introduction

Recently, several distributed infrastructures including Grids and Clouds have been proposed for large-scale collaborative and distributed e-business and e-science applications. We expect the deployment of a large number of Grid and Cloud services for workflow applications such as Cloud workflow systems [1]. Despite the attractive features of these platforms (in terms of scalability, dynamicity and low cost), the inherent unreliability of these open systems has caused great threat to the applications. Resources may be offline unexpectedly, perform unpredictably due to resource sharing between virtualized services, and behave maliciously. Therefore, in large-scale distributed systems, the scheduling of an application must also account for reliability, in addition to execution time (makespan) which is normally the only consideration. To enable reliable scheduling, two important issues need to be considered: (i) how to evaluate the reliability of a resource and (ii) how to perform reliable scheduling based on the reliability information of resources.

Most reliability-oriented scheduling work assumes that the reliability of a resource (denoted as failure rate) is already known. Hence they do not define how to evaluate the reliability of a resource in a widely distributed computing environment. To evaluate the resource reliability, Sonnek et al. [2] and Song et al. [3] employ traditional reputation systems to capture the reliability of resources in their scheduling. This results in two problems. First, from the resource perspective, most reputation models [2, 4, 5, 6] only evaluate the reputation of a resource according to its ratio of successfully completed tasks. They do not consider the influence of the task runtime (size). For example, peer A has a higher task failure rate (task failures per unit time) than peer B, so peer B should have a better reputation. But, traditional reputation models will instead predict a better reputation for peer A when peer A executes more tasks with short runtime and peer B executes more tasks with long runtime. This is because peer A may successfully complete more short tasks than peer B. Second, from the task perspective, existing reputation models assign the same reliability (success probability) [2, 3] to all tasks on a resource based on the reputation of the resource. But, the longer a task runs on an unreliable resource, the lower success probability it should have.

Given the resource reliability information, optimizing both makespan and reliability for a workflow application

*Corresponding author
Email addresses: `xf_wang@nudt.edu.cn` (Xiaofeng Wang), `yeocs@ihpc.a-star.edu.sg` (Chee Shin Yeo), `raj@csse.unimelb.edu.au` (Rajkumar Buyya), `sjs@nudt.edu.cn` (Jinshu Su)

with task dependencies is known to be a NP-hard problem [7]. Hence, many list heuristics have been proposed for this problem. However, most of them attempt to achieve makespan [8, 9, 10] or reliability [2, 7, 11, 12] suboptimal solutions, whose optimality cannot be guaranteed [13]. In contrast, Genetic Algorithms (GAs) can give several satisfying solutions for choice by iterative evolutions over generations of scheduling solutions. Although GA is more time consuming than list heuristics, it is acceptable for applications with long runtime. In addition, the speed of GA can be accelerated by using parallel GA technology [14].

Very few GAs have been proposed so far to enable both makespan and reliability optimized scheduling for workflow applications because of the difficulty to preserve their task dependencies while randomly evolving the solutions. Bi-objective Genetic Algorithm (BGA) [15] is the only GA that we know for this problem, but BGA may give invalid solutions which violate the dependency between tasks. Moreover, most existing GAs [13, 15, 16] evolve the scheduling solutions randomly, which may lead to the slow convergence of the GA. However, GAs can be improved by using heuristics to evolve solutions more intelligently. Yet, not many heuristics are specifically proposed for GAs. Although some two-phase heuristics have been proposed and reported to be more efficient than other heuristics [9], they cannot work with GAs due to their evolution mechanism.

Towards addressing these issues of reliability evaluation and GA scheduling, we extend our previous work [17, 18] to propose the Look-Ahead scheduling algorithm with Reliability-Driven (RD) reputation. To evaluate the reliability of resources, our RD reputation considers the runtime of tasks by using the task failure rate (task failures per unit time) of resources to define the reputation. It also provides a real-time reputation that can be used to evaluate the reliability of each task directly using the exponential failure model. Based on the RD reputation, we propose Look-Ahead Genetic Algorithm (LAGA) to intelligently optimize both makespan and reliability for a workflow application.

LAGA incorporates two novel features: (i) it optimizes the typical GA by a new mutation operator according to our proposed resource priority heuristic; and (ii) it uses a novel evolution and evaluation mechanism – the genetic operators (crossover and mutation) evolve the task-resource mapping for a solution, while the task execution order of a solution is determined in the evaluation step using our new max-min strategy. Our max-min strategy is the first two-phase strategy that can work with GAs based on the proposed dynamic task priority heuristics. By using the max-min strategy, LAGA is able to avoid the invalid solution problem encountered by BGA [15]. Most importantly, LAGA can accelerate the evolution of solutions more intelligently by using our evolution and evaluation mechanism.

The remainder of this paper is organized as follows. Section 2 introduces related work. Section 3 presents the system model and assumptions for the reputation based scheduling problem. Section 4 defines the RD reputation and its calculation algorithm. Section 5 formalizes the RD scheduling problem, while LAGA is presented in Section 6. Performance results are presented in Section 7, followed by the conclusions in Section 8.

2. Related Work

We discuss existing work about reputation calculation, heuristics and GAs used in workflow scheduling, and compare them with our work respectively.

The reliability of a resource can be monitored by its reputation, which can be defined as the probability that the resource can deliver the expected utility service [4]. In P2P systems, EigenTrust [5] and PowerTrust [19] compute the local trust value based on the normalized number of successful transactions. In volunteer computing systems, Sonnek et al. [2] calculate the reliability of a worker resource as the ratio of correct responses, while Zhang and Fang [20] use a Bayesian method to evaluate the reputation of a peer based on the decayed amounts of successfully completed and failed tasks. However, none of these reputation models consider the influence of time. In Grid systems, Song et al. [3] use fuzzy logic to evaluate the reputation. Although their reputation model includes task runtime, they do not specify how the task runtime affects the reputation. Other works [21, 22] evaluate the time-related performance based on resource availability. But, these works examine reliability at the hardware level and thus do not consider the task-level behavior. In addition, most of the above mentioned works do not provide methods to predict the real-time task failure rate of a resource, which is needed for reliable task scheduling. To address these issues, our RD reputation is specifically defined to be time dependent and our reputation calculation algorithm is able to provide real-time failure rate evaluation for a resource.

Given the resource reliability evaluation, many list heuristics [2, 7, 8, 9, 10, 11] have been proposed to optimize makespan or reliability for workflow applications. To optimize the makespan, the Heterogeneous Scheduling Algorithm (HSA) [7] selects a task with a greater task criticalness, which is based on the length of the longest path through the task. To optimize the reliability, Dongarra et al. [11] show that tasks should be assigned to the node with the minimum multiplication value of instruction execution time and reliability. Dogan and Özgüner [12] propose a bi-criteria heuristic rule called RDLS that evaluates the priority of a task-resource mapping according to the task size, task start time, computing power of resource, and reliability cost. The two-phase min-min heuristic is found to be the best tested heuristics for workflow applications [9]. It works as follows: (i) for each task, select its assumed resource which can start the task earliest, and (ii) from all the tasks with the assumed resource, selects the task with the minimum end time to be scheduled. However,

this heuristic cannot be used by GAs because the task-resource mapping is only fixed during the evolution phase. Hence, we propose a new max-min strategy specifically for GAs, which uses a novel task priority heuristic to predict the task criticalness more precisely than the existing heuristics.

GAs can give several satisfying solutions by iterative evolutions over generations of scheduling solutions. To our knowledge, BGA [15] is the only existing GA that can optimize both makespan and reliability for a workflow application. But, BGA may give invalid solutions that violate the dependence between tasks. So far, two methods have been proposed to preserve the task dependence during the evolution phase. First, Corrêa et al. [23] define two partitions, V1 and V2 of the tasks such that there is no dependency from a task in V1 to a task in V2. Second, Wang et al. [13] represent a scheduling solution as two strings: the task-resource mapping string and the task execution order string. Even though both methods can solve the invalid solution problem, they do not take into account the reliability of the workflow application. Furthermore, most existing GAs [16, 15, 13] evolve the scheduling solutions randomly and thus may lead to the slow convergence of the algorithm. On the other hand, our LAGA determines the task execution order of a solution by using our max-min strategy, which ensures that the algorithm does not give invalid solutions. Moreover, LAGA uses a novel evolution and evaluation mechanism that is able to accelerate the evolution of solutions more intelligently.

3. System Model and Assumptions

Table 1 lists all the notations used in this paper. We model a workflow job as a Directed Acyclic Graph (DAG): $Job = (V, E)$. V is the set of nodes $v_i (1 \leq i \leq n)$ which denotes the tasks of the workflow job. E is the set of edges $e(i, j)(1 \leq i < j \leq n)$ which represents the dependence between task v_i and v_j with v_i as the parent task and v_j as the child task. A task with no parent task is called an entry task, while a task with no child task is called an exit task. For each task v_i, its weight $|v_i|$ represents the number of instructions to be executed for the task, which is assumed to be known using compiler technology [11]. Similar to previous work [2, 16, 10], we focus on computationally intensive applications where the communication time between tasks is not modeled. Our future work will extend the application model to include the communication time between tasks.

In our widely distributed computing model as shown in Fig. 1, there is a central Manager Server with four components: (i) Resource Manager, (ii) Task Scheduler, (iii) Task Monitor, and (iv) RD Reputation Manager. Resource Manager acts as a broker for the available computing resources in the system. A computing resource can be a local cluster resource, remote Grid resource or Cloud service provider as used in Cloud workflow systems [1, 24]. Let $R = \{r_1, r_2, \ldots, r_m\}$ be m resources available in the

Table 1: List of Notations

Notation	Definition
v_i	A task in a DAG-modeled application
$e(i, j)$	A dependence between tasks v_i and v_j
$\|v_i\|$	The number of instructions of task v_i
r_i	A resource in the system
γ_i	The unitary instruction execution time of resource r_i
rdr_i	The RD reputation for resource r_i
$M(i)$	The resource to which task v_i is scheduled
s_i	The start time of an interval for resource r_i
e_i	The end time of an interval for resource r_i
rt_i	The total CPU time donated by resource r_i in the current time interval
c_i	The number of task failures experienced by resource r_i in the current time interval
$rdr_i^{t_i}$	The recorded RD reputation for resource r_i in time interval t_i
t_i^{avail}	The available start time for task v_i
$idle(r_j)$	The time when resource r_j is idle
t_i^s	The start time of task v_i
t_i^e	The end time of task v_i
t_S^j	The time when resource r_j completes all its tasks in scheduling S
R_S	The success probability of an application in scheduling S
$fail(S)$	The failure factor of scheduling S
$time(S)$	The makespan of scheduling S
$imprt(i)$	The length of the longest path starting from task v_i in a DAG graph
$p(i)$	The priority of task v_i given by heuristics

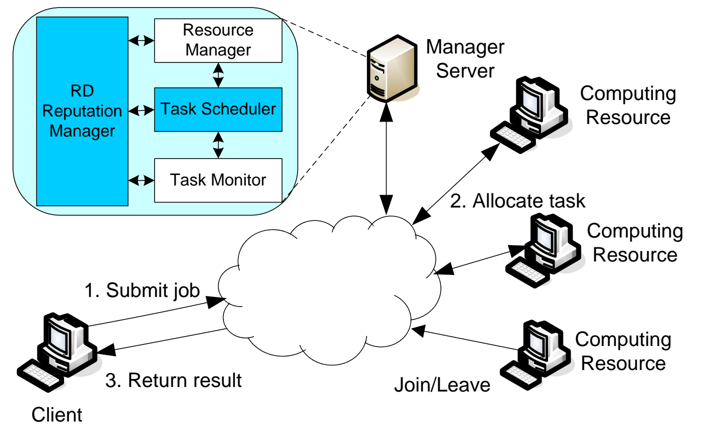


Figure 1: System Model

system. Each resource r_i is associated with two values: (i) γ_i, its computing speed illustrated by unitary instruction execution time (i.e. the time taken to execute one instruction) and (ii) rdr_i, its RD reputation. The reputation rdr_i depicts the failure frequency of resource r_i and is maintained by RD Reputation Manager. Our calculation algorithm for real-time RD reputation will be introduced in Section 4.

With the information of workflow applications and available resources, Task Scheduler can schedule the application. Let $M : V \to R$ denote the mapping function, thus $M(i) = r_j$ means task v_i is assigned to resource r_j. We assume that the central Manager Server can only schedule at most one task to one resource at any time. The scheduling problem and scheduling algorithm will be introduced in Section 5 and 6 respectively.

After schedules are made, Task Scheduler will send tasks to their assigned resources for execution. Meanwhile, it notifies Task Monitor of the schedule information. We assume that Task Monitor can check the status of all tasks. Several technologies have been proposed for Task Monitor such as checkpoint and quizzes verification [25]. Task Monitor can detect if a task v_i successfully completes or fails before completion and sends a reputation report about resource $M(i)$ to RD Reputation Manager. With the collected reputation feedbacks, RD Reputation Manager can maintain a real-time reputation for each resource, which can be used to schedule the next application.

RD Reputation Manager, Task Scheduler and Task Monitor form a self-closed system. With the RD reputation given by RD Reputation Manager, Task Scheduler can schedule workflow jobs to different resources. According to the scheduling information, Task Monitor will keep track of the task execution and give feedbacks to RD Reputation Manager, where the new real-time reputation is calculated for the next workflow scheduling. These three components work iteratively and interactively to maintain real-time reputation prediction and efficient workflow scheduling.

4. Reliability Evaluation as Reputation

In volunteer computing, many discrete events may lead to failures of an application such as non-availability of required services, overloaded resource conditions and malicious activities. All these events are independent and may happen randomly. Hence we use the commonly used exponential distribution [11, 7, 15] to model the failure of a resource provider. The failure density function is $f(t) = \lambda e^{-\lambda t} (t \geq 0)$, where λ is the failure rate of a resource. Let num_fails be the number of failures within a resource during the job runtime period of run_time. We can then compute the failure rate λ, which is the inverse of Mean Time To Failure (MTTF), as:

$$\lambda = \frac{1}{\int_0^\infty \lambda x e^{-\lambda x} dx} = \frac{1}{MTTF} = \frac{num_fails}{run_time} \qquad (1)$$

To enable reliable scheduling, the real-time failure rate of each resource needs to be monitored. Although traditional reputation systems can be used to monitor the reliability of a resource, they neither predict the failure rate of a resource directly nor consider the influence of time. Hence, we define a time-dependent reputation called Reliability-Driven (RD) reputation, which is directly related to the failure rate, as:

Definition 1. The RD reputation rdr_i for a resource r_i is the generally said or believed probability of task failure per unit time, whereby the resource provider will fail to complete its assigned tasks.

4.1. Calculation of Real-time RD Reputation

Algorithm 1 shows the calculation of real-time RD reputation based on successive time intervals, each lasting a time window T_{window}. During each time interval, the server maintains a reputation statistic $repu_sta_i = (s_i, e_i, rt_i, c_i)$ for each resource r_i, where s_i is the start time of the interval, e_i is the end time of the interval, rt_i is the total CPU time donated by resource r_i for task execution in the interval, and c_i is the number of task failures experienced by resource r_i in the interval.

Algorithm 1: RD Reputation Calculation Algorithm

```
1  foreach resource r_i do
2      rdr_i = rdr_i^0 = rdr^initial;
3      t_i ← 1;
4      s_i = e_i = current_time;
5      rt_i ← 0;
6      c_i ← 0;
7  while there is a reputation record testimony_j^i do
8      if e_j^i < s_i + T_window then
           //current interval
9          c_i ← c_i + c_j^i;
10         rt_i ← rt_i + (e_j^i − s_j^i);
11         e_i ← max(e_j^i, e_i);
12         remove record testimony_j^i;
13         compute rdr_i using Equation 2;
14     else
           //next interval
15         rdr_i^{t_i} ← rdr_i;
16         t_i ← t_i + 1;
17         s_i = e_i = s_i + T_window;
18         rt_i ← 0;
19         c_i ← 0;
```

The algorithm begins with initializing the reputation statistic $repu_sta_i$ of each resource r_i for the first time interval (line 1–6). Let us now assume that the algorithm comes to a time interval t_i for resource r_i. After a task v_j assigned to resource r_i successfully completes or fails, the server gives a reputation report $testimony_j^i = (s_j^i, e_j^i, c_j^i)$, where s_j^i and e_j^i are the start and end times of task v_j respectively, and c_j^i is the number of failures during this task. If a task fails, we simply set c_j^i to be 1, otherwise it is 0. The server then uses this report to update the reputation statistic $repu_sta_i$ for resource r_i (line 9–11).

After each update of the reputation statistic $repu_sta_i$, a real-time statistical failure rate for resource r_i can be calculated. Here, the entire length of the current time interval is $e_i - s_i$. During the donated task execution time rt_i of resource r_i in the current interval, resource r_i has c_i task failures. During the remaining time $e_i - s_i - rt_i$ in the current interval, resource r_i is assumed to use the reputation observed in the previous time interval $t_i - 1$. Hence the assumed number of task failures for the remaining time in the current interval is $rdr_i^{t_i-1}(e_i - s_i - rt_i)$, where $rdr_i^{t_i-1}$ is the recorded RD reputation for resource r_i in the previous time interval $t_i - 1$. We can now derive the real-time RD reputation rdr_i for resource r_i by Equation 2. The $(e_i - s_i - rt_i)/(e_i - s_i)$ can be explained as the time decay factor for the recorded RD reputation for resource r_i in the previous time interval.

$$\begin{aligned} rdr_i &= \frac{c_i + rdr_i^{t_i-1}(e_i - s_i - rt_i)}{e_i - s_i} \\ &= \frac{rt_i}{e_i - s_i} \cdot \frac{c_i}{rt_i} + \frac{e_i - s_i - rt_i}{e_i - s_i} \cdot rdr_i^{t_i-1} \end{aligned} \tag{2}$$

At the end of the current time interval t_i, the real-time RD reputation rdr_i for resource r_i is recorded as $rdr_i^{t_i}$ (line 16). The server then starts another reputation statistic for the next time interval $t_i + 1$ (line 17–20). For the initial time interval, we assume that the RD reputation rdr_i^0 for each resource r_i is $rdr^{initial}$ (line 2). $rdr^{initial}$ is the initial RD reputation for all the resources and should be set to a relatively high failure rate. In this way, $rdr^{initial}$ provides resource providers with the incentive to deliver good quality services so as to improve their reputation.

5. Reliability-driven Scheduling Problem

Based on RD reputation, we can define our reliability-driven scheduling problem. In a workflow application, each task can only be executed after all its parent tasks have been completed. Thus the available start time t_i^{avail} for a task v_i is:

$$t_i^{avail} = \max_{e(j,i)\in E} t_j^e \tag{3}$$

where t_j^e is the end time of task v_j. If task v_i has no parent tasks, its available start time is 0. Let function $idle(r_j)$ be the time when resource r_j is idle. Then the start time t_i^s and end time t_i^e of task v_i is defined as:

$$t_i^s = \max\{t_i^{avail}, idle(M(i))\} \tag{4}$$

$$t_i^e = t_i^s + |v_i|\gamma_j \tag{5}$$

where $M(i) = r_j$. $M(i)$ is the resource to which task v_i is assigned and γ_j is the instruction speed of resource r_j. Hence the time t_S^j when resource r_j completes all its assigned tasks in scheduling S is defined as:

$$t_S^j = \max_{i|M(i)=r_j}\{t_i^e\} \tag{6}$$

The reliability of a workflow application is the probability that all its tasks complete successfully. This can be given by the probability that all the resources remain functional until all the tasks assigned to them are completed [11]. Since rdr_i represents the failure rate of resource r_i, the probability that resource r_i can successfully complete all its tasks in scheduling S is $R_S^i = e^{-t_S^i \cdot rdr_i}$. Thus the success probability R_S of an application in scheduling S can be computed as the product of all R_S^i as:

$$R_S = \prod_{i=1}^{m} R_S^i = e^{-\sum_{i=1}^{m} t_S^i \cdot rdr_i} \tag{7}$$

The reliability-driven scheduling of a workflow application is to maximize the reliability and minimize the makespan of the application. To maximize the reliability of scheduling S, we need to minimize its failure factor $fail(S) = \sum_{i=1}^{m} t_S^i \cdot rdr_i$. Therefore the scheduling problem can be formalized as:

$$\text{Minimize } fail(S) = \sum_{i=1}^{m} t_S^i \cdot rdr_i \tag{8}$$

$$\text{Minimize } time(S) = \max_{r_i \in R}(t_S^i) \tag{9}$$

6. Reliability-driven Scheduling using GA

For the scheduling of workflow applications, GAs can give several satisfying solutions for choice by iterative evolutions over generations of scheduling solutions. A typical GA consists of the following steps: (1) create an initial population consisting of randomly generated solutions (chromosomes); (2) evaluate the fitness of each solution and select the solutions for the next population; (3) generate a new generation of solutions by applying two genetic operators (crossover and mutation); and (4) repeat step 2 and 3 until the population converges. Usually, the genetic operators evolve a scheduling solution randomly [16, 15, 13], which can give invalid solutions or lead to slow convergence of the algorithm. To address this problem, we design the Look-Ahead Genetic Algorithm (LAGA) which uses a novel evolution and evaluation mechanism. LAGA determines the task execution order for a solution in the evaluation step instead of normally in the evolution step (crossover and mutation), and uses a new max-min strategy based on our proposed task priority heuristics. Each step of LAGA is explained further in the following sections.

6.1. Encoding

For a workflow application, a chromosome is a data structure in which a scheduling solution is encoded. As illustrated in Fig. 2(c), we use a two-dimensional string to represent a scheduling solution. One dimension of the string represents the index of resources, which depicts the task-resource mapping, while another dimension denotes

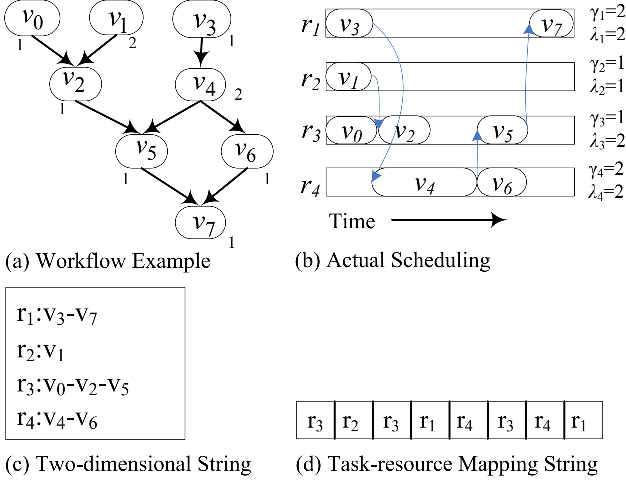


Figure 2: Encoding

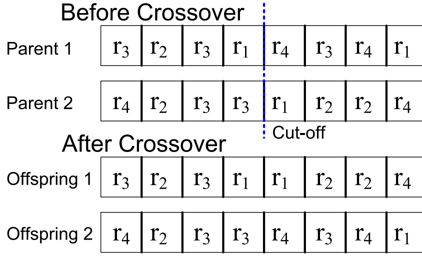


Figure 3: Crossover

the order between tasks. The two-dimensional string can then be converted into the task-resource mapping string M (Fig. 2(d)) directly, which is a vector of length $|V|$. Note that the task-resource mapping string has the same symbol M as the mapping function since they mean the same, i.e. $M(i) = r_j$ means task v_i is assigned to resource r_j.

6.2. Crossover

The rationale for the crossover operation is that it may result in an even better chromosome by exchanging two fittest chromosomes. To keep the dependence between tasks, two crossover operations have previously been designed to exchange task-resource mapping and task execution order separately and randomly [16, 13]. But this can make it difficult for the crossover operation to find a better solution since a good task execution order for one task-resource mapping does not necessarily mean it is also good for another task-resource mapping. Hence our crossover operation only exchanges the task-resource mapping between two chromosomes. The task order of the two new offspring is to be determined later in the evaluation step using our proposed task priority heuristics. This ensures that a feasible task execution order of a specific task-resource mapping is determined more intelligently instead of just random exchange.

Our crossover operation first randomly selects some

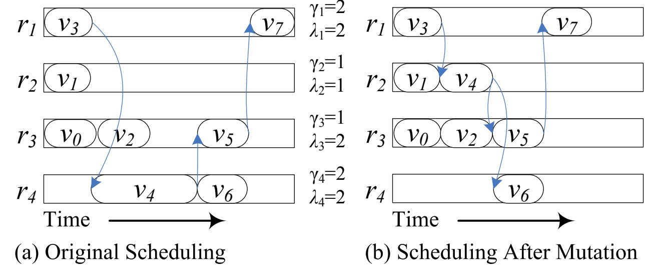


Figure 4: Mutation

pairs of chromosomes from the current population with a probability p_c. For each pair as shown in Fig. 3, it randomly generates a cut-off point for the task-resource mapping string M, which divides the strings of the pair into top and bottom parts. The two bottom parts are then exchanged to generate two new task-resource mapping offspring.

6.3. Mutation

The mutation operation leads the search to exit from a local optimum and typically changes some of the genes in a chromosome randomly. But this may cause the algorithm to search randomly around the good solutions. Therefore, in our mutation operation, a solution is mutated intelligently based on a resource priority heuristic. To optimize the reliability of an application, Dongarra et al. [11] have proven that the resource which has the minimal multiplication value of instruction speed (unitary instruction execution time) and failure rate should have a higher priority to be selected in the scheduling. Hence we define:

Lemma 1. *Resource Priority Heuristic (ResPH): Let $1/(\gamma_i rdr_i)$ be the priority of a resource r_i, and S be a schedule where all the tasks are assigned to the resource with the highest priority. Then any other schedule $S' \neq S$ with reliability $R_{S'}$ is such that $R_{S'} < R_S$.*

Like the crossover operation, our mutation operation only changes the task-resource mapping for a solution. It mutates a solution in the generation with a probability p_m. The mutation operation randomly selects one task in the solution and reassigns it to any resource which has a lower $\gamma_i rdr_i$. As shown in Fig. 4(a), task v_4 is originally scheduled to resource r_4 whose $\gamma_i rdr_i$ is 4, thus the mutation operation reassigns it to resource r_2 with a lower $\gamma_i rdr_i$ of 1. Fig. 4(b) shows the new scheduling, in which both makespan and reliability of the application have been improved.

6.4. Evaluation

Most GAs only assess the fitness of a solution in the evaluation step, but do not try to improve the solution according to the evaluation result. In our evaluation operation, LAGA schedules the task execution order for a new solution first. Then it calculates the estimated end time t_i^e

for each task v_i so that we can evaluate the makespan and failure factor of the new scheduling S using Equation 5.

To give an optimized task execution order for a specific task-resource mapping string, we first define two task priority heuristics, and then propose a new max-min scheduling strategy based on these two heuristics. To optimize the makespan for an application, we need to assign higher priority to tasks which can either start earlier or have more influence on the makespan of the application. Hence we define:

Definition 2. Task Priority Heuristic 1 (TaskPH1): Let the importance $imprt(i)$ of a task v_i be the length of the longest path beginning from the task in the DAG graph, which can be denoted as:

$$imprt(i) = \begin{cases} |v_i| & \text{if } v_i \text{ is an exit task} \\ |v_i| + \max\limits_{e(i,j)\in E} imprt(j) & \text{otherwise} \end{cases} \quad (10)$$

Thus the priority $p(i)$ of task v_i is:

$$p(i) = E(\gamma) \cdot imprt(i) - \max(t_i^{avail}, idle(M(i))) \quad (11)$$

where $E(\gamma)$ is the mean instruction speed of all resources.

If there are two tasks scheduled to the same resource, the one with the higher priority should be scheduled first. TaskPH1 uses the mean instruction speed of all resources to estimate the completion time of the longest path beginning from a task. It is easy and simple to be implemented. In our GA, since the task-resource mapping has been fixed in the previous evolution step, we can have a more precise estimation of the completion time for a path. Hence we define:

Definition 3. Task Priority Heuristic 2 (TaskPH2): Let the estimated completion time $comp(i)$ for the longest path beginning from task v_i be:

$$comp(i) = \begin{cases} |v_i| \cdot \gamma_j & \text{if } v_i \text{ is an exit task} \\ |v_i| \cdot \gamma_j + \max\limits_{e(i,k)\in E} comp(k) & \text{otherwise} \end{cases} \quad (12)$$

where $M(i) = r_j$. Thus the priority $p(i)$ of task v_i is:

$$p(i) = comp(i) - \max(t_i^{avail}, idle(M(i))) \quad (13)$$

Based on the task priority heuristics, our evaluation algorithm uses a two-phase max-min strategy as shown in Algorithm 2, which is specifically proposed for GAs. For each resource, the algorithm first selects its next to be scheduled task which has the maximum priority based on TaskPH1 or TaskPH2. Then, from all the next to be scheduled tasks of the resources, it selects the task with the minimum end time to be scheduled. Given a task-resource mapping string M (the mapping function), all the tasks assigned to resource r_i are put into que_i in their scheduling order, and the algorithm outputs the estimated completion time t_S^i for each resource r_i in the new scheduling S. que_ready_i is the queue containing the unscheduled tasks which are ready to run on resource r_i.

The algorithm works as follows: (i) add each entry task v_j to the task ready queue of its assigned resource $M(j)$ and set its available start time to 0 (line 1–3); (ii) select the task with the maximum priority for each resource (line 8); (iii) among all the selected tasks, the task v_{task_sel} with the minimum end time is selected to be scheduled (line 9–12); (iv) schedule the selected task v_{task_sel} (line 13–15), and update the task completion time and idle time for resource $M(task_sel)$ (line 16); (v) update the state of all the child tasks of the scheduled task (line 17–20); (vi) repeat step ii–v until all the tasks have been scheduled.

Theorem 1. *The time complexity of the evaluation algorithm is $O(n \log n + nm + d)$, where m is the number of resources, n is the number of nodes (tasks) in a DAG, and d is the number of directed edges (dependence constraints).*

Proof. The time complexity of initializing the task ready queue is $O(n)$ (line 1–3). An entire iteration (line 4–21) schedules one task at a time. Thus it will run n times. To effectively sort and select a task for each resource (line 8), it takes $O(\log n)$ time. The time complexity of computing the task end time and select the task with the minimum end time is $O(m)$ (line 9–12). The time complexity of line 13–16 is $O(1)$. Hence the time complexity of repeating line 5–16 is $O(n(\log n + m + 1))$. To update the available time for the child tasks (line 17–20), it takes O(d) time. Therefore, the total time complexity for the evaluation algorithm is $O(n + n(\log n + m + 1) + d) = O(n \log n + nm + d)$.

6.5. Selection

In GAs, the fitness function is used to measure and select solutions. As our goal is to optimize the reliability and makespan for an application under the time constraint, the Sum of Weighted Global Ratios (SWGR) model [15] can be used to compute the fitness value. Thus the fitness value $f(S)$ of a scheduling S is defined as:

$$f(S) = \omega_1 \cdot \frac{fail(S) - minFail}{maxFail - minFail} + \omega_2 \cdot \frac{time(S) - minTime}{maxTime - minTime}, (\omega_1 + \omega_2 = 1) \quad (14)$$

Here, $maxFail$ and $minFail$ are the maximum and minimum failure factors for the solutions in the current generation respectively, while $maxTime$ and $minTime$ are the maximum and minimum makespan respectively. The first two elements of $f(S)$ encourage the algorithm to select the solutions with the minimum failure factor and minimum makespan. Both these two objectives are assigned weights (ω_1 and ω_2) according to the trade-off requirement of the user. Hence, to select the good solutions for the next

Algorithm 2: Evaluation Algorithm

```
   input  : task-resource mapping string M
   output: {t_S^i, que_i} for each resource r_i
1  foreach entry task v_j do
2      add v_j to task ready queue que_ready_M(j);
3      t_j^avail <- 0;
4  repeat
       //minimum end time
5      min_end <- ∞;
       //task selected
6      task_sel <- null;
7      foreach resource r_i do
           //max-min phase 1
8          find task v_j with the maximum priority value from
           que_ready_i;
           //max-min phase 2
9          compute t_j^e using Equation 5;
10         if t_j^e < min_end then
               //update minimum end time
11             min_end <- t_j^e;
               //update task selected
12             task_sel <- j;
13     res_sel <- M(task_sel);
14     remove v_task_sel from que_ready_res_sel;
15     add v_task_sel to que_res_sel;
16     t_S^res_sel = idle(res_sel) = t_task_sel^e;
17     foreach child task v_i of task v_task_sel do
18         compute t_i^avail using Equation 3;
19         if v_i is ready to run then
20             add v_i to que_ready_M(i);
21 until every que_ready_i is empty;
```

generation, the chromosomes in the mating pool are first ordered in the ascending order of their fitness value $f(S)$. After that, the algorithm uses the commonly used roulette wheel selection scheme [13] to select solutions for the next generation. Details of this scheme can be found in [13] and thus not repeated here.

7. Performance Evaluation

We simulate a distributed computing environment to evaluate the performance of our scheduling problem. We first examine the impact of RD reputation on the reputation calculation and the scheduling result. We then assess the performance of LAGA by comparing it with two popular list heuristics and another GA, and analyzing the efficiency of its three priority heuristics.

7.1. Experimental Setup

For our experiments, we use GridSim [26] to simulate a distributed computing environment based on three parameters: the number of resources m, the mean resource speed γ, and the mean resource failure rate λ. These parameters are used according to previous work [8, 7, 15]. There are 200 resources donating various number of CPU cycles. The resource speeds are uniformly distributed between 5×10^{-4} and 10^{-3} milliseconds per instruction. The resource failure rates are uniformly distributed between 10^{-3} and 10^{-4} failures/h [15].

As in most previous work [8, 15, 13, 27], we use a random DAG graph generator to simulate workflow applications based on three parameters: the number of tasks, the mean out-degree of a task node, and the mean task size. The number of tasks is between 40 and 200. The mean out-degree of a task node is 2. The task sizes are uniformly distributed between 1×10^4 and 15×10^6 Million Instructions (MI).

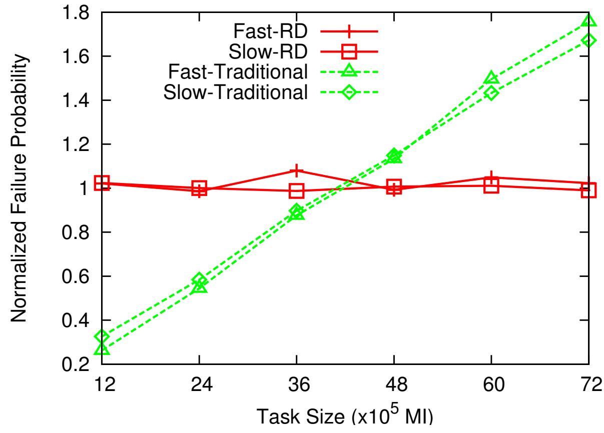
Other remaining parameters are for our proposed RD reputation and LAGA. The initial RD reputation $rdr^{initial}$ for all resources is 10^{-3} failures/h and the reputation decay factor α is 0.2. Both weights ω_1 and ω_2 for the fitness value are 0.5, i.e. the algorithm considers both reliability and makespan to be of the same priority. The probability for crossover operation and mutation operation are 0.5 and 0.25 respectively. These two probabilities are set to be a medium value used in [14] so that we can test the evolution process of GAs. The population size of LAGA is 20. For each type of workflow application with the same parameters, we create 5 instances so that they can have a wide representation. In addition, for each workflow application, we run the GAs 3 times to obtain their average results.

7.2. RD Reputation

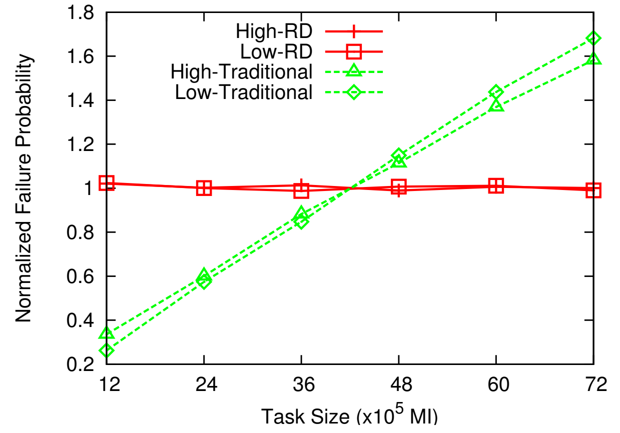
7.2.1. Comparison with Traditional Reputation

We compare our RD reputation with traditional reputation, which uses the ratio of successfully completed tasks. Our comparison focuses on increasing task sizes of $\{12, 24, 36, 48, 60, 72\} \times 10^5$ MI in two scenarios: (i) varying resource speed and (ii) varying resource failure rate. The varying resource speed is either fast with 1000 MIPS or slow with 500 MIPS. The varying resource failure rate is either high with 10^{-3} failures/h or low with 10^{-4} failures/h.

Fig. 5 shows the failure probabilities of RD and traditional reputation for a medium-sized task normalized by the actual failure rate of the resource. The failure probability of RD reputation across different task sizes remain consistently about the same as the actual failure rate of the resource (i.e. normalized value of 1), whereas the failure probability of traditional reputation is only close to the actual failure rate for the medium task size (i.e. 42×10^5 MI). Otherwise, the failure probability of traditional reputation also increases as the size of tasks increases. In addition, the failure probability of traditional reputation deviates more from the correct failure rate when resources have a faster speed (Fig. 5(a)) or lower failure rate (Fig. 5(b)). This is because the normalized failure probability of traditional reputation obeys a negative exponential function. Thus the faster speed or lower failure rate contributes to a smaller exponent which results in a greater deviation.

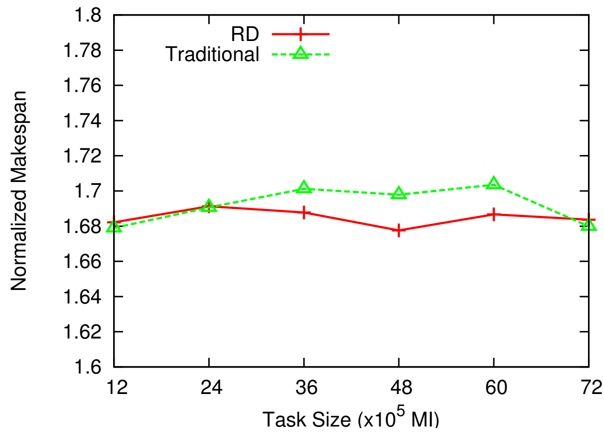


(a) Varying Resource Speed

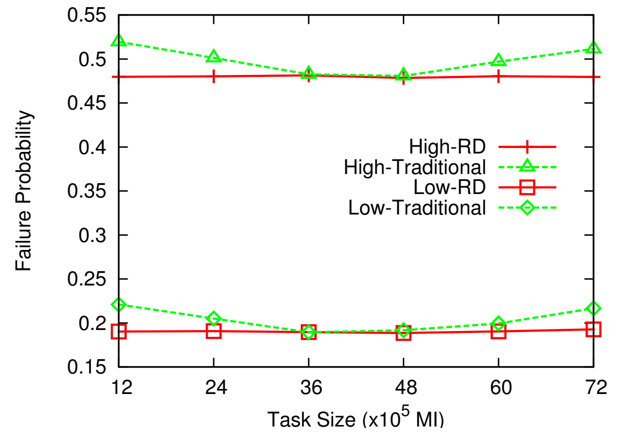


(b) Varying Resource Failure Rate

Figure 5: Normalized failure probability of a medium-sized task based on RD and traditional reputation



(a) Normalized Makespan



(b) Failure Probability

Figure 6: Normalized Makespan and failure probability of a workflow application based on RD and traditional reputation

7.2.2. Impact on Scheduling

To compare the scheduling results based on RD and traditional reputation, half of the resources in our simulation have the actual failure rate, while the other half of the resources have either RD reputation based failure rate or traditional reputation based task failure probability. Fig. 6 shows both RD and traditional reputation based scheduling achieves almost the same makespan for increasing task sizes (Fig. 6(a)). However, RD reputation based scheduling achieves a consistently lower failure probability than traditional reputation based scheduling (Fig. 6(b)). In particular, the failure probability of traditional reputation is much worse than RD reputation as the task size decreases or increases. This is due to the traditional reputation giving a different resource failure rate from the actual one, and hence tasks are scheduled to more unreliable resources.

7.3. LAGA

7.3.1. Comparison with List Heuristics

DLS, RDLS [12] and BSA [7] are three well known list heuristics to optimize makespan or reliability for workflow applications. To compare our LAGA with these three list heuristics, we run DLS, RDLS and BSA 100 times respectively to get the average result. The number of tasks varies from 40 to 200. Fig. 7 shows that LAGA performs the best for both makespan and reliability. In particular, LAGA achieves a considerably larger improvement ratio (of about 14%) for makespan and reliability when the number of tasks is small (40 tasks), as compared to when the number of tasks is large (200 tasks). This is because when there are fewer tasks, there will be more idle resources for LAGA to select for each task. Hence LAGA is able to examine each of them to find the most suitable resource. On the other hand, list heuristics only examine one resource according to the heuristic value, which may not be the best one.

7.3.2. Comparison with Another GA (BGA)

We compare our LAGA with BGA [15] which evolves a solution randomly. Their performances are compared in terms of iteration and time. For the comparison in terms of iteration, we compute the average normalized makespan and reliability of the solutions, which are the mean makespan and reliability of the current generation normalized by the mean makespan and failure factor of the initial generation respectively. The application has 200 tasks, while the number of iterations of the two GAs are 1000. Fig. 8 shows that LAGA improves the makespan and reliability for an application more quickly than BGA. In addition, for the same number of iterations, LAGA always gives better quality scheduling solutions than BGA.

To compare the performance in terms of time, we run two experiments. The first experiment examines the average time needed for a new generation with better quality solutions. Fig. 9(a) shows BGA needs less time (16 milliseconds less on average) than LAGA for a new generation at the start of the evolution (generation 1–31), but needs much more time with increasing number of generations (generation 62 onwards). Since BGA randomly searches for better solutions, it is easy to find a better solution at the start of the evolution, but gradually becomes harder to find a better solution. On the other hand, LAGA needs to run the evaluation algorithm which is more time-consuming and thus evolves more slowly at the start of the evolution. However, our priority heuristics are able to ensure that LAGA still continues to find better solutions using the similar amount of time as the evolution process progresses.

The second experiment studies the average scheduling quality over the running time of the GA by sampling every 200 milliseconds. The normalized scheduling quality of a GA is the sum of the normalized reliability and the inverse of the normalized makespan. In Fig. 9(b), at the start of the evolution, BGA improves the quality of the solutions more quickly than LAGA. But, it becomes very difficult and slower for BGA to improve the quality, while LAGA outperforms BGA to obtain better quality solutions. Hence, Fig. 9(b) proves our analysis of the previous experiment (Fig. 9(a)) is correct. Fig. 9(b) also shows that for a workflow application with 200 tasks, it needs about 5.2 seconds for LAGA to achieve convergence. This is acceptable for long runtime workflow applications that need several hours or even several days.

7.4. Priority Heuristics

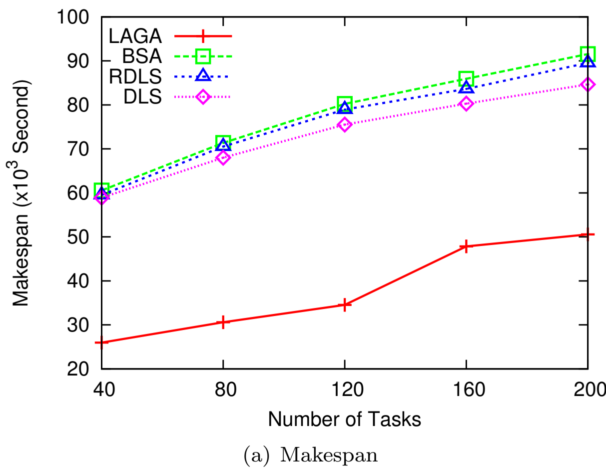
7.4.1. Efficiency of ResPH

To evaluate the efficiency of ResPH, we use a workflow application comprising of 200 tasks to compare LAGA using only ResPH (LAGA-ResPH) with BGA. Fig. 10 shows the average normalized scheduling quality (as introduced in Section 7.3.2) of LAGA-ResPH and BGA over the number of iterations. It can be seen that ResPH gives better scheduling quality at the start of the evolution. But, as the number of iterations increases, ResPH is no longer able to further improve the scheduling quality, thus resulting in LAGA achieving the same scheduling quality as BGA. This shows that after a period of evolution, it will be difficult for LAGA to improve the quality of a solution by just assigning a task to a faster or more reliable resource.

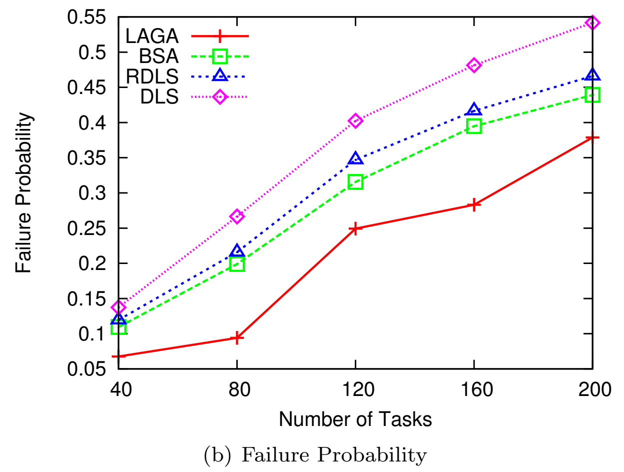
7.4.2. Efficiency of TaskPH1 and TaskPH2

We compare the efficiency of our task priority heuristics (TaskPH1 and TaskPH2) on LAGA. For the experiments, the number of tasks in the workflow application varies from 40 to 200. Fig. 11 shows the average makespan and reliability of the solutions given by LAGA using TaskPH1 and TaskPH2, normalized by the makespan and reliability of the solutions given by BGA respectively.

We can observe that both TaskPH1 and TaskPH2 enable LAGA to achieve a lower makespan (normalized makespan < 1) and a higher reliability (normalized reliability > 1) than BGA. But, TaskPH2 achieves a significantly lower makespan and higher reliability than TaskPH1 when the

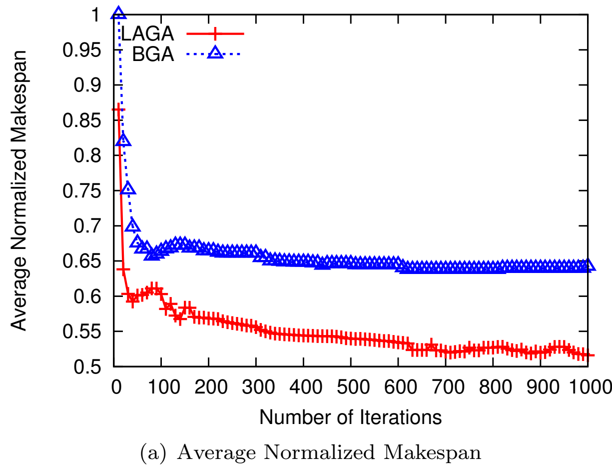


(a) Makespan

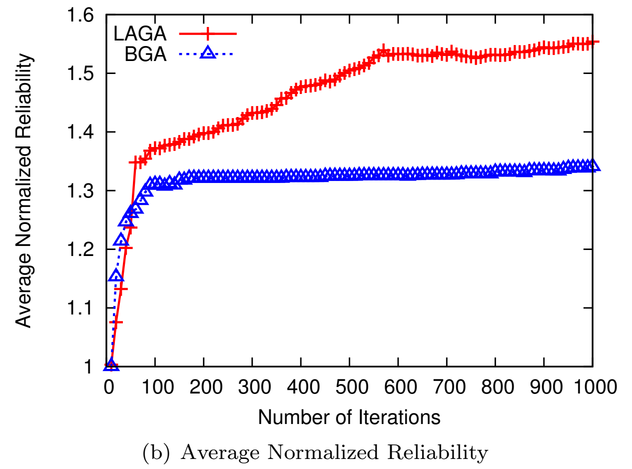


(b) Failure Probability

Figure 7: Makespan and failure probability of a scheduling given by DLS, RDLS and LAGA

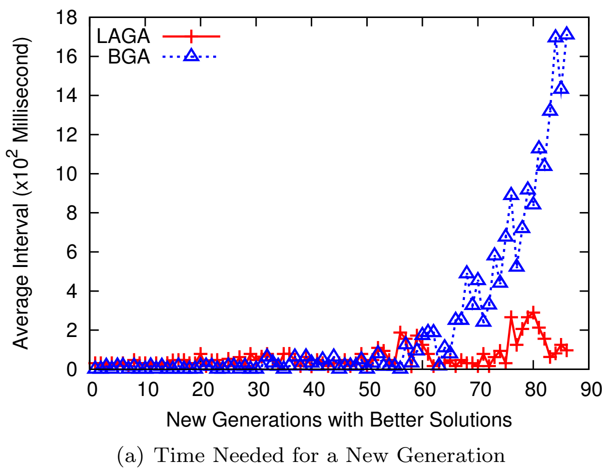


(a) Average Normalized Makespan

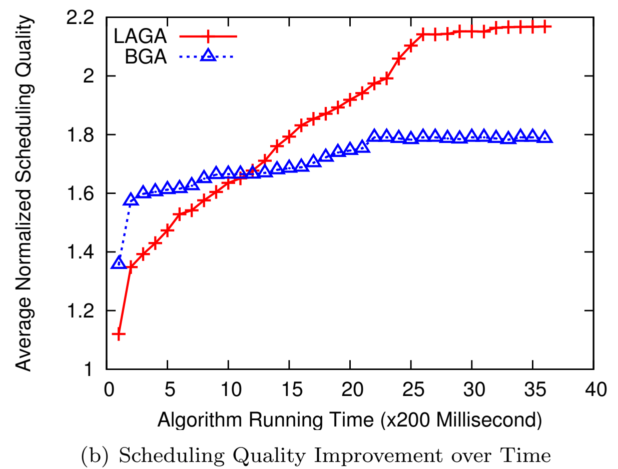


(b) Average Normalized Reliability

Figure 8: Average normalized makespan and reliability of a scheduling given by BGA and LAGA in terms of iterations

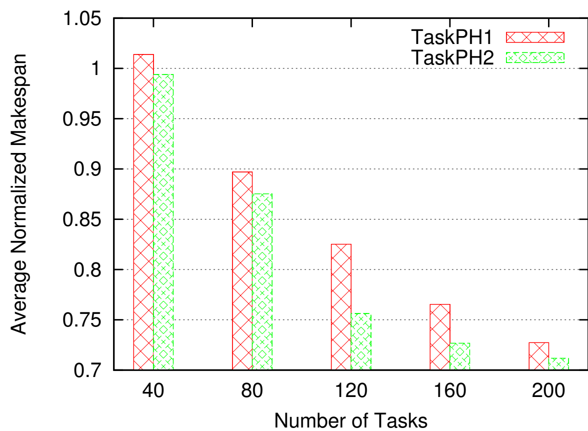


(a) Time Needed for a New Generation

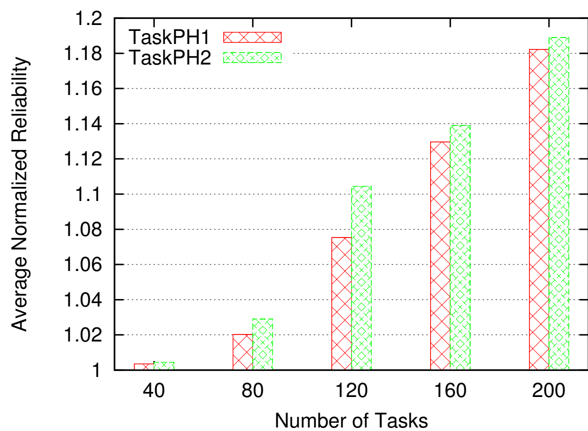


(b) Scheduling Quality Improvement over Time

Figure 9: Performance of BGA and LAGA in terms of time



(a) Average Normalized Makespan



(b) Average Normalized Reliability

Figure 11: Average normalized makespan and reliability of a scheduling given by LAGA using TaskPH1 or TaskPH2

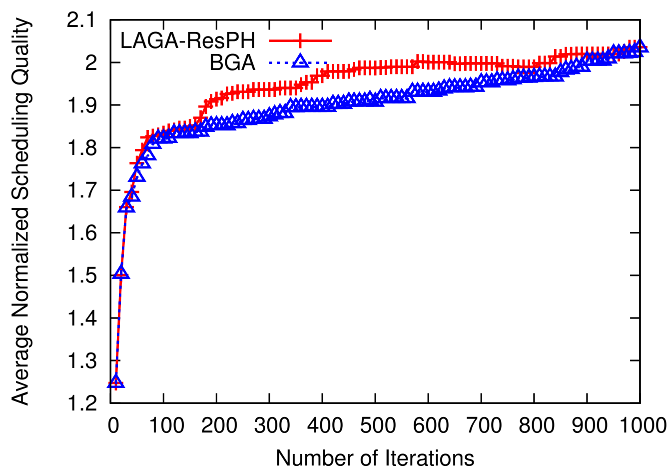


Figure 10: Efficiency of ResPH

workflow application is of medium size (120 tasks). Otherwise, both TaskPH1 and TaskPH2 achieve similar performance when the workflow application is of small or large size. This is because when the number of tasks is small, the GA can find a good solution even without heuristics. When the number of tasks is large, every resource will be assigned many tasks, which makes it very difficult to estimate the completion time for a long task path. Hence in this case, TaskPH2 cannot outperform TaskPH1 although it can predict a more precise estimation of the completion time.

8. Conclusion

We have studied the reliability-driven scheduling problem in distributed computing environments by proposing the time-dependent RD reputation for resource reliability evaluation. The RD reputation uses the failure rate to define the reputation of a resource so that it can be used to evaluate the reliability of a task directly using the exponential failure model. Our RD reputation calculation algorithm can also monitor the real-time changes of the reputation dynamically.

Based on the RD reputation, we then propose a Look-Ahead Genetic Algorithm (LAGA) to optimize both makespan and reliability of workflow applications intelligently. LAGA optimizes the typical GA by a new mutation operator according to our proposed resource priority heuristic (ResPH). It uses a novel evolution and evaluation mechanism: the genetic operators evolve the task-resource mapping for a solution, while the task execution order of a solution is determined in the evaluation step using our proposed max-min strategy based on the defined task priority heuristics (TaskPH1 and TaskPH2). Simulation results show that the RD reputation can improve the reliability of a workflow application with more accurate reputations, while LAGA can derive much better quality solutions than list heuristics DLS, RDLS and BSA, and outperforms the previously proposed BGA in evolving scheduling solutions.

Acknowledgements

We thank Marco A. S. Netto and Sungjin Choi for their comments. The work is partially supported by the National Natural Science Foundation of China (Research on Trust Management for Cyber Space) and the Australian Research Council.

References

[1] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Grid Computing Environments Workshop 2008 (GCE 2008), IEEE Computer Society, Austin, TX, USA, 2008.

[2] J. Sonnek, A. Chandra, J. Weissman, Adaptive reputation-based scheduling on unreliable distributed infrastructures, IEEE Trans. Parallel Distrib. Syst. 18 (11) (2007) 1551–1564. doi:http://dx.doi.org/10.1109/TPDS.2007.1094.

[3] S. Song, K. Hwang, Y.-K. Kwok, Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling, IEEE Trans. Comput. 55 (6) (2006) 703–719. doi:http://dx.doi.org/10.1109/TC.2006.89.

[4] A. Jøsang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, Decis. Support Syst. 43 (2) (2007) 618–644. doi:http://dx.doi.org/10.1016/j.dss.2005.05.019.

[5] S. D. Kamvar, M. T. Schlosser, H. Garcia-Molina, The eigentrust algorithm for reputation management in p2p networks, in: WWW '03: Proceedings of the 12th international conference on World Wide Web, ACM, New York, NY, USA, 2003, pp. 640–651. doi:http://doi.acm.org/10.1145/775152.775242.

[6] Y. Wang, M. P. Singh, Trust representation and aggregation in a distributed agent system, in: AAAI '06: Proceedings of the 21st national conference on Artificial intelligence, AAAI Press, 2006, pp. 1425–1430.

[7] M. Hakem, F. Butelle, Reliability and scheduling on systems subject to failures, in: ICPP '07: Proceedings of the 2007 International Conference on Parallel Processing, IEEE Computer Society, Washington, DC, USA, 2007, p. 38. doi:http://dx.doi.org/10.1109/ICPP.2007.72.

[8] S. C. Kim, S. Lee, J. Hahm, Push-pull: Deterministic search-based dag scheduling for heterogeneous cluster systems, IEEE Trans. Parallel Distrib. Syst. 18 (11) (2007) 1489–1502. doi:http://dx.doi.org/10.1109/TPDS.2007.1106.

[9] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R. F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.

[10] M. Wieczorek, S. Podlipnig, R. Prodan, T. Fahringer, Bi-criteria scheduling of scientific workflows for the grid, in: 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), IEEE Computer Society, Washington, DC, USA, 2008, pp. 9–16. doi:http://dx.doi.org/10.1109/CCGRID.2008.21.

[11] J. J. Dongarra, E. Jeannot, E. Saule, Z. Shi, Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems, in: SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures, ACM, New York, NY, USA, 2007, pp. 280–288. doi:http://doi.acm.org/10.1145/1248377.1248423.

[12] A. Dogan, F. Özgüner, Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 308–323. doi:http://dx.doi.org/10.1109/71.993209.

[13] L. Wang, H. J. Siegel, V. R. Roychowdhury, A. A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel Distrib. Comput. 47 (1) (1997) 8–22. doi:http://dx.doi.org/10.1006/jpdc.1997.1392.

[14] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, B.-S. Lee, Efficient hierarchical parallel genetic algorithms using grid computing, Future Gener. Comput. Syst. 23 (4) (2007) 658–670. doi:http://dx.doi.org/10.1016/j.future.2006.10.008.

[15] A. Doğan, F. Özgüner, Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems, Comput. J. 48 (3) (2005) 300–314. doi:http://dx.doi.org/10.1093/comjnl/bxh086.

[16] J. Yu, M. Kirley, R. Buyya, Multi-objective planning for workflow execution on grids, in: GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, IEEE Computer Society, Washington, DC, USA, 2007, pp. 10–17. doi:http://dx.doi.org/10.1109/GRID.2007.4354110.

[17] X. Wang, R. Buyya, J. Su, Reliability-oriented genetic algorithm for workflow applications using max-min strategy, in: 9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2009), IEEE Computer Society: Los Alamitos, CA, USA, Shanghai, China, 2009.

[18] X. Wang, C. S. Yeo, R. Buyya, J. Su, Reliability-driven reputation based scheduling for public-resource computing using ga, in: IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA 2009), IEEE Computer Society: Los Alamitos, CA, USA, Bradford, UK, 2009.

[19] R. Zhou, K. Hwang, Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing, IEEE Trans. Parallel Distrib. Syst. 18 (4) (2007) 460–473. doi:http://dx.doi.org/10.1109/TPDS.2007.1021.

[20] Y. Zhang, Y. Fang, A fine-grained reputation system for reliable service selection in peer-to-peer networks, IEEE Transactions on Parallel and Distributed Systems 18 (2007) 1134–1145. doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2007.1043.

[21] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, H. Casanova, Characterizing resource availability in enterprise desktop grids, Future Gener. Comput. Syst. 23 (7) (2007) 888–903. doi:http://dx.doi.org/10.1016/j.future.2006.11.001.

[22] X. Qin, T. Xie, An availability-aware task scheduling strategy for heterogeneous systems, IEEE Trans. Comput. 57 (2) (2008) 188–199. doi:http://dx.doi.org/10.1109/TC.2007.70738.

[23] R. C. Corrêa, A. Ferreira, P. Rebreyend, Scheduling multiprocessor tasks with genetic algorithms, IEEE Trans. Parallel Distrib. Syst. 10 (8) (1999) 825–837. doi:http://dx.doi.org/10.1109/71.790600.

[24] M. Rahman, R. Ranjan, R. Buyya, Cooperative and Decentralized Workflow Scheduling in Global Grids, Future Generation Computer Systems 26 (5) (2010) 753–768.

[25] S. Zhao, V. Lo, Result Verification and Trust-based Scheduling in Peer-to-Peer Grids, in: Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P 2005), IEEE Computer Society: Los Alamitos, CA, USA, Konstanz, Germany, 2005. doi:10.1109/P2P.2005.13.

[26] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, R. Buyya, A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim, Concurrency and Computation: Practice and Experience 20 (13) (2008) 1591–1609. doi:10.1002/cpe.1307.

[27] M. Bubak, T. Gubala, M. Kapalka, M. Malawski, K. Rycerz, Workflow Composer and Service Registry for Grid Applications, Future Generation Computer Systems 21 (1) (2005) 79–86.