2009 International Conference on Advanced Information Networking and Applications

# Reliability-Driven Reputation Based Scheduling for Public-Resource Computing Using GA

Xiaofeng Wang[#1], Chee Shin Yeo[*1], Rajkumar Buyya[*2], Jinshu Su[#2]

*#College of Computer, National University of Defence Technology*
*Changsha, 410073, Hunan, China*
*{xf_wang[1], sjs[2]}@nudt.edu.cn*

**GRIDS Laboratory, Department of Computer Science and Software Engineering*
*The University of Melbourne, VIC 3010, Australia*
*{csyeo[1], raj[2]}@csse.unimelb.edu.au*

***Abstract*—** For an application in public-resource computing environments, providing reliable scheduling based on resource reliability evaluation is becoming increasingly important. Most existing reputation models used for reliability evaluation ignore the time influence. And very few works use a robust genetic algorithm to optimize both time and reliability for a workflow application. Hence, in this paper, we propose the reliability-driven (RD) reputation, which is time dependent and can be used to evaluate a task's reliability directly using the exponential failure model. Based on the RD reputation, we also propose Knowledge-Based Genetic Algorithm (KBGA) to optimize both time and reliability for a workflow application. KBGA uses heuristics to accelerate the evolution process without giving invalid solutions. Our experiments show that the RD reputation can improve the reliability of a workflow application with more accurate reputation, while the KBGA can evolve to better scheduling solutions more quickly than traditional genetic algorithms.

***Keywords*—** reliability, reputation, workflow scheduling, genetic algorithm, heuristic

## 1. Introduction

Public-resource computing which combines elements of Peer-to-Peer (P2P) and Grid computing is an important technology, and is used in many applications such as SETI@Home and BOINC [3]. Usually, public-resource computing comprises a large number of unsupervised resources which have no prior trust and are more susceptible to unreliability. Hence, many factors may lead to failures for an application. For example, a resource may be overloaded, slow connected, misconfigured or malicious. Thus, in public-resource computing, the scheduling of an application must also account for reliability, besides execution time (makespan) which is normally considered. To enable reliable scheduling, two important issues need to be considered: (i) how to evaluate a resource's reliability and (ii) how to perform reliable scheduling based on the resource's reliability information.

Reputation systems are commonly used to evaluate a resource's reliability [1,2,9,11,13]. But, most existing reputation systems have two problems. Firstly, from the resource perspective, most reputation models [1,2,9,11] evaluate a resource's reputation according to its ratio of successfully completed tasks. They do not consider the influence of the task's 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 short runtime tasks and peer B executes more tasks with longer runtime. This is because peer A may complete more short tasks successfully than peer B. Secondly, from the task perspective, existing reputation models assigned the same reliability (success probability) [1,13] to all the tasks on a resource based on the resource's reputation. But, the longer a task runs on an unreliable resource, the lower success probability it should have.

Given the resource reliability information, it is known to be a NP-hard problem to optimize both makespan and reliability for a workflow application with task dependencies [19]. Several list heuristics have been proposed for this problem in non-genetic algorithms [7,15,16]. Usually, genetic algorithms (GAs) can provide better quality solutions than list heuristics [6,12]. Although GA is more time consuming, it is acceptable for applications with long runtime. Moreover, the speed of GA can be accelerated by adopting parallel genetic algorithm technology [14]. Currently, bi-objective genetic algorithm (BGA) [17] is the only GA that we know can give both makespan and reliability optimized scheduling solutions for workflow applications. But BGA may give invalid solutions which violate the dependence between tasks. In addition, most GAs [8,10,17] evolve the scheduling solutions randomly, which may lead to slow convergence of the algorithm.

In this paper, we propose the novel reliability-driven (RD) reputation model for resource reliability evaluation. RD reputation considers the runtime of tasks by using the resource's task failure rate (task failures per unit time) to define the reputation. It also provides a real time reputation that can be used to evaluate a task's reliability directly using the exponential failure model. Based on RD reputation, we then define the reliability-driven scheduling problem and two scheduling heuristics which aim to optimize makespan and reliability for a workflow application. Finally, we design the knowledge-based genetic algorithm (KBGA) to provide scheduling solutions. KBGA evolves the task execution order according to the task's importance value, so that the scheduling will not violate the dependency between tasks. The mutation of KBGA has two operators namely swapping mutation and reassigning mutation which evolve the solutions intelligently based on our heuristics.

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

## 2. Related Work

The real time resource reliability can be monitored by the resource's reputation, which can be defined as the probability that the resource can deliver the expected utility service [2]. For P2P systems, two popular reputations EigenTrust [9] and PowerTrust [11] were designed. They compute the local trust value based on the normalized number of successful transactions between two participants. For public-resource computing systems, Sonnek et al. [1] calculated a worker's reliability as the ratio of correct responses. Neither the normalized number nor the ratio of correct responses considered the time influence. Song et al. [13] used fuzzy logic to evaluate the reputation. Although task runtime is included in their model, they did not specify how the task runtime affects the reputation. The time related performance can also be evaluated by the resource availability [4]. But it focused on the hardware analysis, not including the task level behaviour analysis. Moreover, most existing works did not give methods or algorithms to predict the real time task failure rate for a resource, which is needed for task scheduling. However, our reliability-driven reputation is specially defined to be time dependent, and our reputation calculation algorithm can provide a real time failure rate evaluation for a resource.

Optimizing both makespan and reliability for a workflow application is known to be a NP-hard problem. Many list heuristics have been proposed [7,15,16]. Dongarra et al. [15] proved that tasks should be scheduled to the node with the minimum multiplication value of the instruction execution time $\gamma$ and reliability $\lambda$. Marek et al. [7] proposed a general bi-criteria scheduling heuristic which divides the scheduling into primary and secondary scheduling. Generally, a genetic algorithm can give better scheduling solutions than list heuristics [12]. Dogan et al. proposed a bi-objective genetic algorithm (BGA) for workflow applications [17]. BGA evolves the scheduling solutions randomly which may give invalid solutions violating the dependency between tasks. Wang et al. [10] represented a scheduling solution as two strings: the task-resource assignment string and the task execution order string. Although this method can solve the invalid solution problem, they did not consider reliability. Most existing GAs [8,10,17] also evolve the scheduling solutions randomly, which may lead to slow convergence of the algorithm. In contrast, our KBGA evolves the task execution order according to the task's importance value and mutates a solution based on our two heuristics. Thus, our KBGA can evolve the scheduling solutions intelligent without giving invalid solutions.

## 3. Models and Assumptions

In the typical public-resource computing model [1] as shown in Fig. 1, there is a central server which assigns jobs submitted by the clients to the resource providers. We model a workflow job as a Directed Acyclic Graph (DAG): $Job = (V, E)$ . $V$ is the set of nodes $v_i (1 \le i \le n)$ which denotes the tasks of the workflow job. E is the set of edges $e(i, j)(1 \le i < j \le n)$ which represents the dependence between tasks $v_i$ and $v_j$, $v_i$ is the parent task and $v_j$ is the child task. For each task node $v_i$, its weight $|v_i|$ is the number of instructions of this task which is assumed to be known using compiling technology [15]. The length of a path in the DAG is the sum of the weights of all nodes along the path.

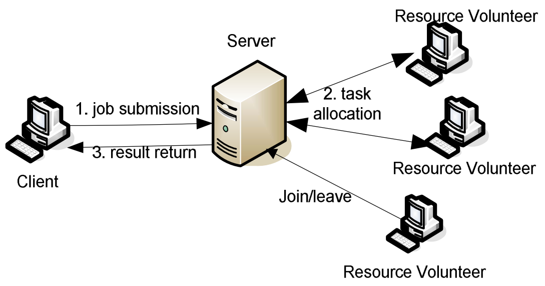


Fig. 1. System Model.

There are some resource volunteers in the system, which are not centrally controlled and will join or leave the system dynamically. Let $R = \{r_1, r_2 \cdots r_m\}$ be the $m$ resources available in the system. Each resource $r_i$ is associated with two values: $rdr_i$, the resource's RD reputation and $\gamma_i$ , the resource's computing speed illustrated by unitary instruction execution time (i.e. the time to execute one instruction). Given the resource's information, the central server can schedule the workflow job. Let $M : V \to R$ denotes the mapping function, and then $M(i) = r_j$ means task $v_i$ is assigned to resource $r_j$.

We assume that the central server can only schedule at most one task to one resource at any time. We also assume that the central server can monitor the task execution, hence if a task successfully finishes or fails before completion, the server can detect it and send a reputation report. Several technologies have been proposed to deal with this problem such as checkpoint and quizzes verification [18].

## 4. Reliability Evaluation as Reputation

In public-resource 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 Poisson Distribution [15,16,17] to model the failure of a resource provider. The failure density function is $f(t) = \lambda e^{-\lambda t} (t \geq 0)$, where $\lambda$ 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 compute the failure rate by Equation 1 which is the inverse of Mean Time To Failure (MTTF).

$$\lambda = \frac{1}{\int_0^\infty \lambda x e^{-\lambda x} dx} = \frac{1}{MTTF} = \frac{num\_fails}{run\_time}. \qquad (1)$$

To enable reliable scheduling, the resource's real time failure rate should be monitored. Although traditional reputation systems can be used to monitor the resource's reliability, they neither predict the failure rate for a resource directly nor consider the time influence. Here, our time dependent reputation is directly related to the failure rate, which can be defined as:

***Reliability-Driven (RD) Reputation ($rdr_i$)*** of a resource $r_i$ is the generally said or believed probability of task failure per unit time, with which the resource provider will fail to complete the tasks assigned to it.

### *4.1 Calculation of Real time RD Reputation*

A resource's RD reputation represents its real time failure rate $\lambda$ introduced above. To maintain the RD reputation, we divide the successive time into time intervals which last a window time $T_{window}$. For each time interval, the server maintains a reputation statistic $repu\_sta_i = (s_i, f_i, runtime_i, c_i)$ for each resource $r_i$. The variables $s_i$ and $f_i$ are the start and finish times for an interval respectively, $runtime_i$ is the total CPU time that resource $r_i$ donates for task execution in the interval, and $c_i$ is the number of failures experienced by tasks. Algorithm 1 shows the RD reputation calculation algorithm. It begins with initializing each resource's reputation statistic $repu\_sta_i$ for the first time interval (line 1~6).

Let us assume that the algorithm comes to time interval $t_i$ for resource $r_i$. After a task $v_j$ assigned to resource $r_i$ successfully finishes or fails, the server gives a reputation report $testimony_j^i = (s_j^i, f_j^i, c_j^i)$, where $s_j^i$ and $f_j^i$ are the start and finish times of task $v_j$ respectively, and $c_j^i$ is the number of failures during this task. If a task fails, we simply assign $c_j^i$ to be 1, otherwise it is 0. The server uses this report to update the reputation statistic $repu\_sta_i$ ( line 9~11).

After each update of the reputation statistic $repu\_sta_i$, a real time statistical failure rate $\lambda_i^{statistic}$ for resource $r_i$ can be computed using Equation 1. Here, the whole length of the current time interval is $f_i - s_i$. During the $runtime_i$ of the resource's donated task execution time in the current interval, the resource has $c_i$ task failures. During the remaining time $f_i - s_i - runtime_i$ in the current interval, the resource is assumed to work with a reputation observed in the last time interval $t_i - 1$. Thus the assumed number of task failures for the remaining time in the current interval is $rdr_i^{t_i - 1}(f_i - s_i - runtime_i)$, where $rdr_i^{t_i-1}$ is the recorded RD reputation for resource $r_i$ in the last time interval $t_i - 1$. And we can get the real time statistic failure rate by:

$$\lambda_i^{statistic} = \frac{c_i + rdr_i^{t_i - 1}(f_i - s_i - runtime_i)}{f_i - s_i}. \qquad (2)$$

The reputation should decay over time, thus the real time RD reputation for resource $r_i$ in the current time interval $t_i$ can be defined as:

$$rdr_i = \alpha \cdot rdr_i^{t_i - 1} + (1 - \alpha)\lambda_i^{statistic}, (0 \leq \alpha < 1) \quad (3)$$

where $\alpha$ is the decay factor. If $\alpha$ is zero, the real time RD reputation will be equal to $\lambda_i^{statistic}$, which means it is totally decided by the current statistical failure rate.

At the end of the current time interval $t_i$, the real time RD reputation $rdr_i$ is recorded as $rdr_i^{t_i}$ for resource $r_i$ (line 16), and the server starts another reputation statistic for the next time interval $t_i + 1$ (line 17~19). 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. It should be set to a relatively high failure rate. In this way, it gives resource providers incentives to supply good quality services to improve their reputation.

**Algorithm 1** RD Reputation Calculation Algorithm

```
1  for each resource $r_i$ do
2      $rdr_i = rdr_i^0 = rdr^{initial}$
3      $t_i \leftarrow 1$
4      $s_i = f_i = current\_time$
5      $runtime_i \leftarrow 0;\ c_i \leftarrow 0$
6  end for
7  while there is a reputation record $testimony_j^i$ do
8      if ($f_j^i < s_i + T_{windows}$) then      //current interval
9          $c_i \leftarrow c_i + c_j^i$
10         $runtime_i \leftarrow runtime_i + (f_j^i - s_j^i)$
11         $f_i \leftarrow \max(f_j^i, f_i)$
12         Remove the record $testimony_j^i$
13         Compute $\lambda_i^{statistic}$ by Equation 2
14         Compute $rdr_i$ by Equation 3
15     else                                      //next interval
16         $rdr_i^{t_i} \leftarrow rdr_i$
17         $t_i \leftarrow t_i + 1$
18         $s_i = f_i = s_i + T_{windows}$
19         $runtime_i \leftarrow 0;\ c_i \leftarrow 0$
20     end if
21 end while
```

## 5. Reliability-Driven Scheduling Problem

In this section, the reliability-driven scheduling problem based on RD reputation is formalized first. Then two heuristics are defined for genetic algorithms to improve the scheduling solutions.

### *5.1 Problem Representation*

In a workflow application, each task could be executed only after all its parent tasks have been completed. Thus the available start time for a task $v_i$ is:

$$t_i^{avail} = \max_{e(j,i)\in E} t_j^e , \tag{4}$$

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

$$\begin{gathered} t_i^b = \max\{t_i^{avail}, idle(M(i))\} \\ t_i^e = t_i^b + |v_i|\gamma_j \quad where\ M(i) = r_j \end{gathered} , \tag{5}$$

where $M(i)$ is the resource to which task $v_i$ is assigned, and $\gamma_j$ is the instruction speed of resource $r_j$. Let $t_S^j$ be the time when resource $r_j$ finishes all the tasks assigned to it in scheduling *S*, which can be 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. It can be given by the probability that all the resources remain functional until all the tasks assigned to it are completed [15]. Since $rdr_i$ represents the failure rate for 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 $R_S$ for an application in scheduling *S* can be computed as the product of all $R_S^i$, which is illustrated in Equation 7. We can see that to maximize the reliability, we need to minimize the failure factor $fal(S) = \Sigma_{i=1}^m t_S^i rdr_i$.

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

The reliability-driven scheduling of a workflow application is to maximize the reliability and minimize the makespan for the application within the time constraint of the deadline *D*. Therefore the scheduling problem can be formalized as:

$$\begin{gathered} \text{Minimize} \quad fal(S) = (\sum_{i=1}^m t_S^i \cdot rdr_i) \\ \text{Minimize} \quad time(S) = \max_{r_i \in R}(t_S^i) \\ \text{Subject to} \quad time(S) < D \end{gathered} . \tag{8}$$

### *5.2 Heuristic rules*

To maximize the reliability, Heuristic 1 can be applied [15]. It has been proved that to maximize the reliability, the task should be scheduled to the resource with minimal $\gamma_i rdr_i$ whenever it is possible.

**Heuristic 1**

Let $S$ be a schedule where all the tasks are assigned to a resource with minimum $\gamma_i rdr_i$ . Then any schedule $S' \neq S$ with reliability of $R_{S'}$ is such that $R_{S'} < R_s$ .

To minimize the makespan for an application, we should give higher priority to tasks that can start earlier and to tasks that have a bigger influence to the makespan of the application. Thus the second heuristic can be defined as:

**Heuristic 2**

Let the importance 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:

$$impt(i) = \begin{cases} |v_i| & \forall j, e(i,j) \notin E \\ |v_i| + \max\limits_{e(i,j)\in E} impt(j) & otherwise \end{cases} . \qquad (9)$$

And the task $v_i$ 's priority $p(i)$ is:

$$p(i) = E(\gamma) \cdot impt(i) - \max(t_i^{avail}, idle(M(i))) \ , \qquad (10)$$

where $E(\gamma)$ is the mean instruction speed of all resources. Then, if there are two tasks scheduled to the same resource, the one with the higher priority should be scheduled first.

## 6. Reliability-driven Scheduling using Genetic Algorithm

For the scheduling problem of workflow applications, a GA can usually give better solutions than list heuristics [12]. A typical GA consists of the following steps: (1) create an initial population consisting of randomly generated solutions which are also called chromosomes; (2) evaluate the fitness of each solution and remove poor solutions from the population; (3) generate a new generation of solutions by applying two evolution operators, namely crossover and mutation; and (4) repeat step 2 and 3 until the population converges. In order to make a GA converge to better solutions more quickly without giving invalid solutions, we design the knowledge-based genetic algorithm (KBGA). KBGA evolves the task execution order according to the task's importance value. It also optimizes the typical GA by applying two new mutation operators based on our two heuristics. The details of KBGA are presented in the following sections.

### *6.1 Chromosome Encoding and Crossover*

For workflow applications, a chromosome is a data structure into which a scheduling solution is encoded. We use a two-dimensional encoding string [8] to represent a scheduling solution. As illustrated in Fig. 2c, one dimension of the string represents the index of resources, while the other dimension shows the order of tasks on each resource. The two-dimensional string can be converted into a one-dimensional string according to the resource's index and task's order. The one-dimensional string comprises a list of ordered pairs (i, j), also called a gene. The pair (i, j) denotes task $v_i$ is scheduled to resource $r_j$. The order between tasks in the one-dimensional string only makes sense when tasks are scheduled to the same resource.

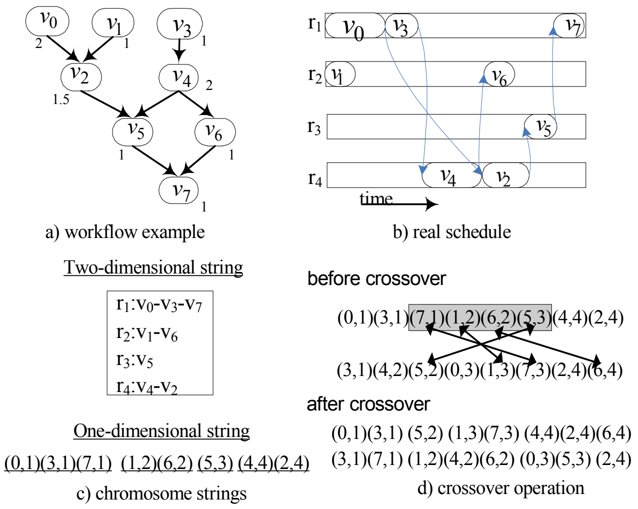


Fig. 2. Encoding and Crossover Example.

The crossover operation creates new chromosomes by randomly exchanging part genes of the existing chromosomes. As illustrated in Fig. 2d, our algorithm performs the crossover operation on the one-dimensional string as follows: (1) Two chromosomes are randomly chosen from the current population, and two random genes are selected from one of the chromosomes; (2) All the genes between the selected two genes are chosen as crossover genes, and the resource allocation for all the tasks related to the crossover genes are exchanged between the selected two chromosomes; and (3) For each resource in the two new chromosomes, the tasks assigned to it are rescheduled in the descending order of their importance value $impt(i)$ . In this way, the parent tasks are always scheduled before their child tasks, thus avoiding the invalid solution problem [17]. After crossover, two new offspring are generated by combining task assignments taken from the two parents.

### *6.2 Mutation*

Typically, a mutation operation changes some of the genes in a chromosome randomly, which causes the algorithm to search randomly around the good solutions. We obtain two new mutation operators, namely *reassigning mutation* and *swapping mutation*. They use the two defined heuristics to help the algorithm evolve more directly to the good solutions.

The reassigning mutation improves the reliability for a scheduling using *Heuristic 1*. First, it chooses a task in one scheduling solution randomly. Then it reassigns the task to a resource with a lower $\gamma_i rdr_i$ , and schedules the task order according to its importance value $impt(i)$. In Fig. 3a, task $v_6$ is originally scheduled to resource $r_2$ whose $\gamma_i rdr_i$ is 2. The reassigning mutation reassigns it to resource $r_1$ with a lower $\gamma_i rdr_i$ of 1 as shown in Fig. 3b. Hence the reliability of the workflow application has been improved, although the makespan remains the same.

The swapping mutation improves the makespan for a scheduling according to *Heuristic 2*. It randomly chooses a resource in one scheduling, and compares the priority of two successive tasks on the resource. It swaps the execution order of the two tasks if the preceding task has a lower priority. In Fig. 3a, task $v_4$ is scheduled before $v_2$ because it has a higher importance value, but has a lower priority. Therefore the swapping mutation exchanges their execution order. Fig. 3c shows the new scheduling where the makespan of the application has been reduced.

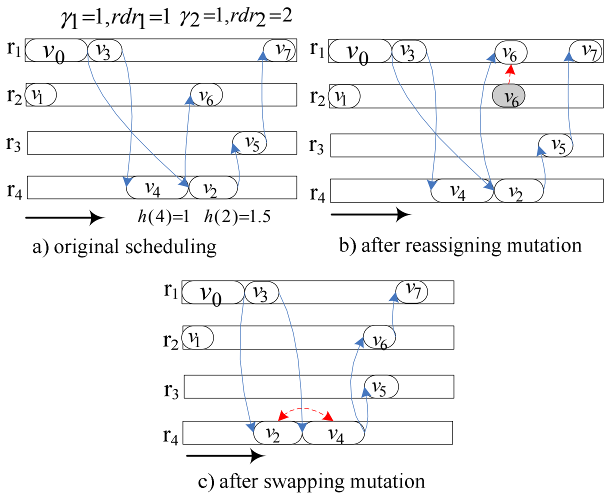

$\gamma_1=1, rdr_1=1$ $\gamma_2=1, rdr_2=2$
r1
r2
r3
r4
$v_0$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$h(4)=1$ $h(2)=1.5$
a) original scheduling
b) after reassigning mutation
c) after swapping mutation

Fig. 3. Mutation Operation.

## *6.3 Evaluation*

In the evolutionary-based optimization methods, fitness functions are used to measure the quality of a solution according to the optimization objectives. As our goal is to optimize the reliability and makespan for a workflow application under the time constraint, the fitness value $f(s)$ for a scheduling solution $S$ can be defined as:

$$f(s)=\omega_1\cdot\frac{fal(s)-\text{minFal}}{\text{maxFal}-\text{minFal}}+\omega_2\cdot\frac{time(s)-\text{minTime}}{\text{maxTime}-\text{minTime}} + f_{penalty}(s) \qquad (\omega_1+\omega_2=1) \quad . \quad (11)$$

$$\text{where} \quad f_{penalty}(s)=\begin{cases}0 & \text{if} \quad time(s)<D\\ 1 & \text{if} \quad time(s)>D\end{cases}$$

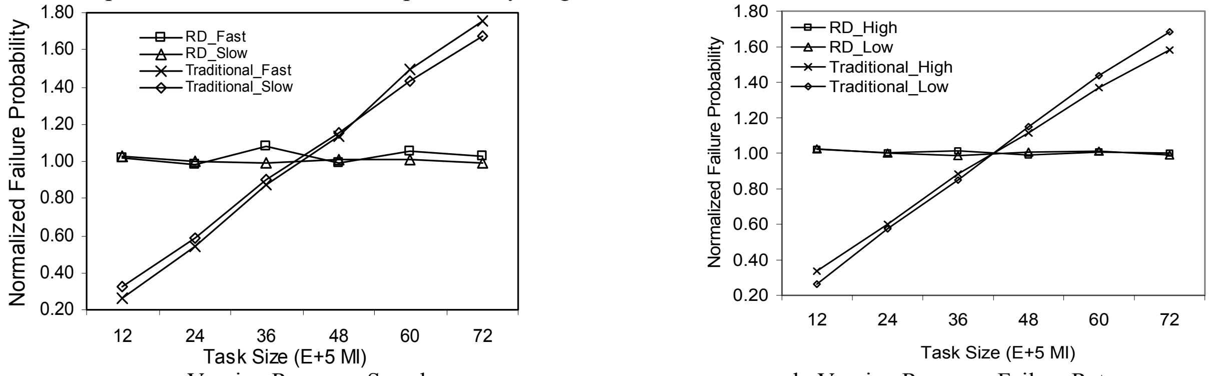
Here, maxFal and minFal are the maximum and minimum failure factors for the solutions in the current population respectively, while maxTime and minTime are the maximum and minimum makespan respectively. The first two elements of $f(s)$ encourage the algorithm to choose the solutions with minimum failure factor and minimum makespan. Both these two objectives are assigned a weight according to the user's trade-off requirement. The third element $f_{penalty}(s)$ is to handle the time constraint. If the makespan of a scheduling exceeds the time deadline D, the function will give a penalty to its fitness value.

## 7. Experiments

We use GridSim [5] to simulate a public-resource computing environment for our experiments. There are 200 resource providers in the system. They donate various numbers of CPU cycles whose speed is uniformly distributed in $[5\times10^{-4}, 10^{-3}]$ milliseconds per instruction. The actual failure rates for resource providers are assumed to be uniformly distributed from $10^{-3}/h$ ψto $10^{-4}/h$ [17]. The structure of a workflow application can be categorized into balanced and unbalanced [8]. Like other previous works [8,16,17], we use a random DAG generator to simulate the application. Our simulated workflow application consists of 300 tasks. The mean outdegree for a task node is 2. The task's size is chosen uniformly between $5\times10^3$ Million instructions (MI) and $72\times10^5$ MI. The reputation decay factor is 0.2, while the fitness evaluation weight $\omega_1$ and $\omega_2$ are both set to be 0.5 so that the algorithm gives the same priority to both reliability and makespan.

**a) RD reputation compared with traditional reputation:** The traditional reputation model uses the ratio of successfully completed tasks as a resource's reputation. To compare the difference between RD and traditional reputations, we test the two reputations under several extreme conditions: the size of all the test tasks in the system are $\{12,24,36,48,60,72\}$ $\times10^5$MI respectively, the resource provider has a high failure rate of $10^{-3}/h$ or a low failure rate of $10^{-4}/h$, and the resource provider donates resources of a fast speed of 1000MIPS or a slow speed 500MIPS. To facilitate the comparison, we derive the task failure probabilities for a medium-sized task based on the traditional and RD reputation. Fig. 4 shows the two failure probabilities normalized by the standard task failure probability based on the resource's actual failure rate. The task failure probabilities based on RD reputation remain consistently close to the standard task failure probability. The traditional reputation based task failure probability gets close to the standard task failure probability only when the test tasks in the system also have the medium task size. Otherwise, the failure probability also increases as the size of the test tasks increases. And when the resources have a faster speed (Fig. 4a) or lower failure rate (Fig. 4b), the task failure probability based on traditional reputation will have a greater deviation from the correct one. This is because the normalized failure probability based on traditional reputation obeys a negative exponential function. The lower failure rate and faster speed will contribute to a smaller exponent which results in greater deviation.

**b) RD reputation's influence to scheduling:** To compare the scheduling results based on traditional and RD reputation, half of the resources in the 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. 5 shows both the traditional reputation based scheduling and RD reputation based scheduling have almost the same makespan under various conditions. The RD reputation based scheduling also has a consistently lower failure probability, while the traditional reputation based scheduling has a higher failure probability, especially when the reputations are computed under conditions when the task's size is very small or very large. This is because under such conditions, the traditional reputation gives a different resource failure rate from the standard one, and tasks are scheduled to more unreliable resources.



a. Varying Resource Speed

b. Varying Resource Failure Rate

Fig. 4. Normalized failure probability of a medium-sized task based on traditional reputation and RD reputation.

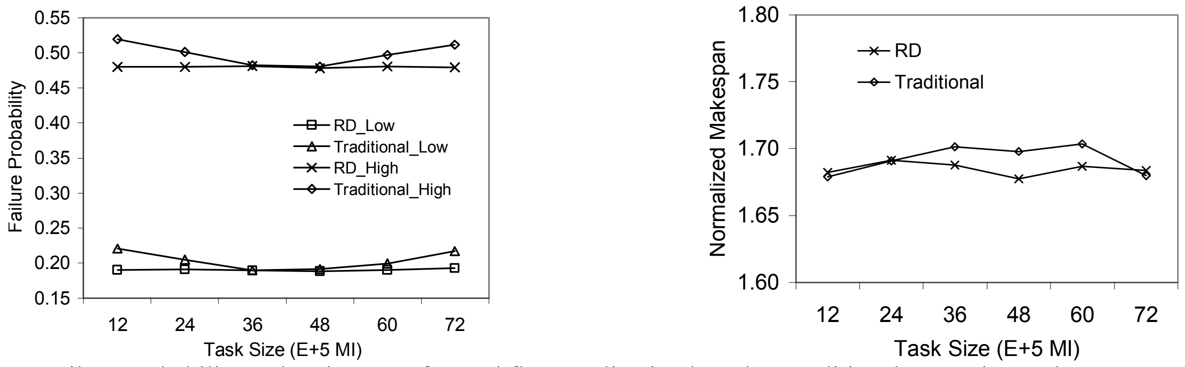


Fig. 5. Failure probability and makespan of a workflow application based on traditional reputation and RD reputation.

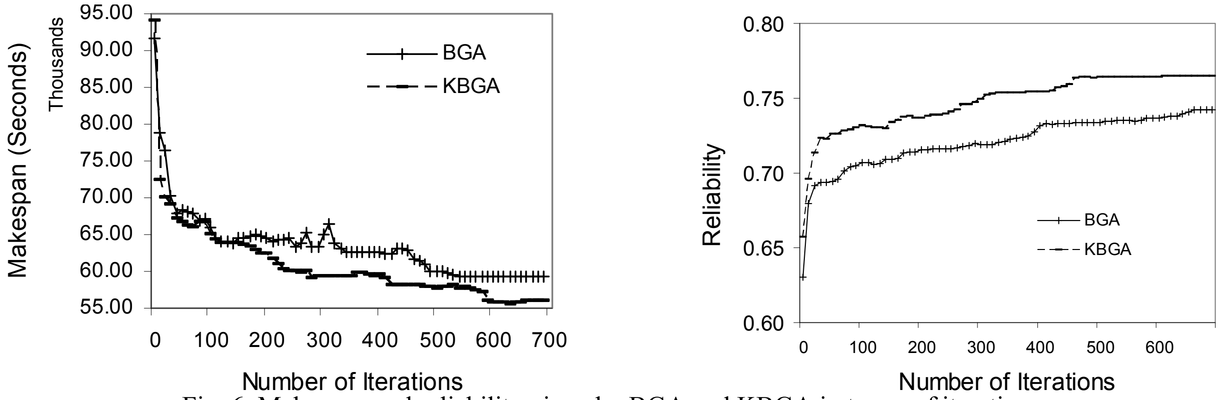


Fig. 6. Makespan and reliability given by BGA and KBGA in terms of iterations.

**c) KBGA's performance:** We compare KBGA with BGA [17] which also optimizes makespan and reliability for an application by evolving solutions randomly. The average makespan and reliability of all the solutions are computed after each iteration. Fig. 6 shows KBGA improves the makespan and the reliability for an application more quickly than BGA. After some iterations, it becomes very difficult for BGA to find a better solution by randomly evolving solutions, while it is easier for KBGA to evolve with heuristics. At the end of the algorithm, KBGA can give a better quality solution than BGA, in particular, the heuristics of KBGA can optimize reliability more than makespan.

## 8. Conclusions

In this paper, we studied the reliability-driven scheduling problem in public-resource computing environments. We proposed the time-dependent RD reputation for resource reliability evaluation. The RD reputation uses the failure rate to define a resource's reputation so that it can be used to evaluate a task's reliability 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 defined the reliability-driven scheduling problem and two heuristics that aim to optimize makespan and reliability for a workflow application. We proposed the KBGA to evolve the scheduling solutions intelligently using the heuristics. KBGA addresses the invalid solution problem by evolving the order between tasks according to their importance value. Simulation results show that the RD reputation model can improve the reliability of a workflow application with more accurate reputation. The KBGA algorithm also outperforms the typical GA in evolving scheduling solutions.

## Acknowledgments

Xiaofeng's visit to the GRIDS Lab at the University of Melbourne is supported by the Chinese Scholarship Council. We thank Marco A. S. Netto and Sungjin Choi for their comments.

## References

[1] J. Sonnek, A. Chandra, and J. Weissman. Adaptive Reputation-Based Scheduling on Unreliable Distributed Infrastructures. IEEE Transactions on Parallel and Distributed Systems, 18(11):1151-1564, 2007.

[2] A. Jøsang, R. Ismail, and C Boyd. A Survey of Trust and Reputation Systems for Online Service Provision. Decision Support Systems, 43(2):618-644, 2007.

[3] I. Foster, and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. 2nd Int'l. Workshop on P2P Systems, 2003.

[4] D. Kondo, G. Fedak, F. Cappello, A.A. Chien, and H. Casanova: Characterizing resource availability in enterprise desktop grids. Future Generation Comp. Syst. 23(7):888-903, 2007.

[5] A. Sulistio, G. Poduval, R. Buyya, and C. Tham, On Incorporating Differentiated Levels of Network Service into GridSim, Future Generation Computer Systems (FGCS), 23(4):606-615, 2007.

[6] X. Wang, R. Buyya and J. Su, Reliability-Oriented Genetic Algorithm for Workflow Applications Using Max-Min Strategy, 9th IEEE International Symposium on Cluster Computing and the Grid, 2009.

[7] M. Wieczorek, S. Podlipnig, R. Prodan, and T. Fahringer. Bi-criteria Scheduling of Scientific Workflows for the Grid. IEEE Symposium on Cluster Computing and the Grid, May, 2008.

[8] J. Yu, M. Kirley, and R. Buyya, Multi-objective Planning for Workflow Execution on Grids, IEEE/ACM Conference on Grid Computing, 2007.

[9] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The Eigentrust Algorithm for Reputation Management in P2P Networks," ACM World Wide Web Conf. (WWW '03), May, 2003.

[10] L. Wang, H. J. Siegel, V. P. Roychowdhury, et al., Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel Distrib. Comput. 47(1):8-22, 1997.

[11] R. Zhou and K. Hwang, "PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing, IEEE Trans. on Parallel and Distributed Systems, 18(5):460-473, 2006.

[12] T. D. Braun, H. J. Siegel, N. Beck et al, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. of Parallel and Distributed Computing, 61(6):810-837, 2001.

[13] S. Song, K. Hwang, and Y.K. Kwok, Risk-Resilient Heuristics and Genetic Algorithms for Security-Assured Grid Job Scheduling, IEEE Trans. on Computers, 55(6):703-719, 2006.

[14] D. Lim, Y. Ong, Y. Jin, B. Sendhoff, B. Lee, Efficient Hierarchical Parallel Genetic Algorithms using Grid computing, Future Generation Computer Systems, 23(4):658-670, 2007.

[15] J. Dongarra, E. Jeannot, E. Saule, and Z. Shi. Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems. ACM Symp. on Parallelism in Algorithms and Architectures 2007.

[16] M. Hakem, and F. Butelle, Reliability and Scheduling on Systems Subject to Failures. International Conference on Parallel Processing(ICPP), Sept. 2007.

[17] A. Dogan and F. Ozguner. Bi-objective Scheduling Algorithms for Execution Time-Reliability Trade-off in Heterogeneous Computing Systems. The Computer Journal. 48(3):300-314, 2005.

[18] S. Zhao and V. Lo, Result Verification and Trust-based Scheduling in Open Peer-to-Peer Cycle Sharing systems, IEEE Conference on Peer-to-Peer Systems, Sept. 2005.

[19] R. Duan, R. Prodan, and T. Fahringer, Performance and Cost Optimization for Multiple Large-scale Grid Workflow Applications, ACM/IEEE Conference on Supercomputing, November, 2007