Risk-Resilient Heuristics and Genetic Algorithms for Security-Assured Grid Job Scheduling

Shanshan Song, Kai Hwang, *Fellow*, *IEEE*, and Yu-Kwong Kwok, *Senior Member*, *IEEE*

Abstract—In scheduling a large number of user jobs for parallel execution on an open-resource Grid system, the jobs are subject to system failures or delays caused by infected hardware, software vulnerability, and distrusted security policy. This paper models the risk and insecure conditions in Grid job scheduling. Three risk-resilient strategies, *preemptive, replication*, and *delay-tolerant*, are developed to provide security assurance. We propose six risk-resilient scheduling algorithms to assure secure Grid job execution under different risky conditions. We report the simulated Grid performances of these new Grid job scheduling algorithms under the NAS and PSA workloads. The relative performance is measured by the total job makespan, Grid resource utilization, job failure rate, slowdown ratio, replication overhead, etc. In addition to extending from known scheduling heuristics, we developed a new *space-time genetic algorithm* (STGA) based on faster searching and protected chromosome formation. Our simulation results suggest that, in a wide-area Grid environment, it is more resilient for the global job scheduler to tolerate some job delays instead of resorting to preemption or replication or taking a risk on unreliable resources allocated. We find that delay-tolerant Min-Min and STGA job scheduling have 13-23 percent higher performance than using risky or preemptive or replicated algorithms. The resource overheads for replicated job scheduling are kept at a low 15 percent. The delayed job execution is optimized with a delay factor, which is 20 percent of the total makespan. A Kiviat graph is proposed for demonstrating the quality of Grid computing services. These risk-resilient job scheduling schemes can upgrade Grid performance significantly at only a moderate increase in extra resources or scheduling delays in a risky Grid computing environment.

Index Terms—Grid computing, job scheduling heuristics, genetic algorithm, replication scheduling, risk resilience, NAS and PSA benchmarks, performance metrics, distributed supercomputing.

1 Introduction

Realistic platforms for Grid computing are facing security threats from network attacks and system vulnerability. Computational Grid is often used to execute a large number of user jobs at dispersed resource sites. Some of the jobs may be dispatched to multiple machine sites for distributed parallel execution. Thus, job outsourcing becomes a major incentive in collaborative Grid computing. Specifically, in a large-scale computational Grid, distributed resource clusters work at different *autonomous domains* (ADs), as depicted in Fig. 1a. Job executions are carried out across the domain boundaries, meaning that the jobs from one AD could be outsourced to another AD for faster execution [6], [31].

However, a major hurdle of such job outsourcing is caused by network security threats [2]. If a Grid site is under attack, its resources may not be accessible from outside of the domain. Thus, a job dispatched to that site may be delayed or failed after system infections leading to machine crashes. To enable more effective job scheduling, it is desirable to know a priori the *security demand* (SD) from Grid jobs, as shown in Fig. 1b and the *trust level* (TL) assured by a resource provider at the Grid site.

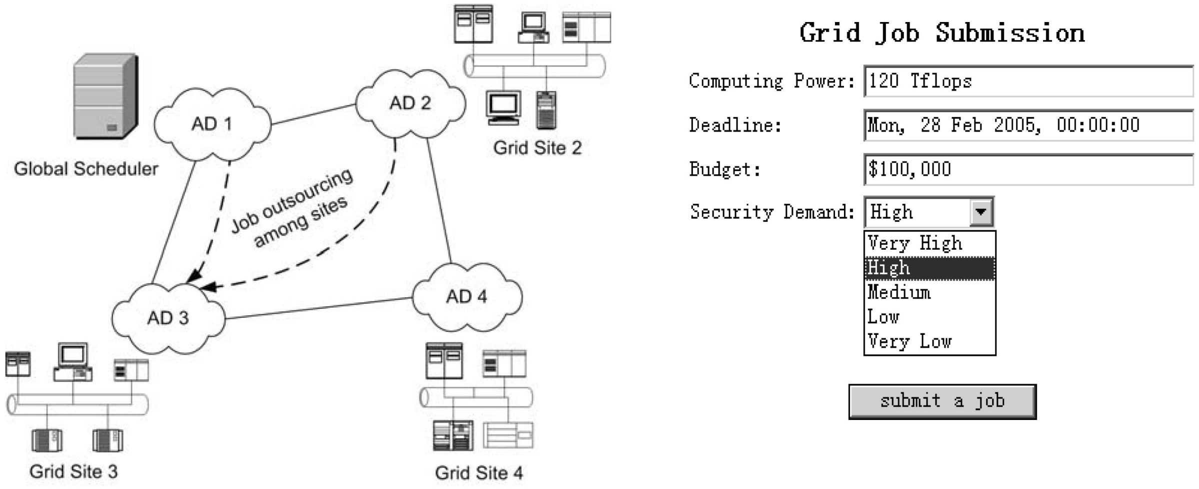
In a real life situation, asking the Grid users to fully specify their SD is an unreasonable burden. This situation is illustrated with a simple Grid job submission request in Fig. 1b. In addition to the request for computing power under deadline and budget limits, the user wishes to simply express an *SD* level from high to low. Obviously, the scheduling of jobs has to take the *risk* factor into account. A practical Grid job scheduler must be *security-driven* and *resilient* in response to all risky conditions. The scheduler must consider the risks involved in dispatching jobs to remote sites. Furthermore, risk-resistant strategies are needed to properly manage the risks it may take. However, despite the fact that many heuristics have been suggested for large-scale job scheduling [8], [10], previously proposed heuristics were not applicable in a risky environment.

In this paper, we have tackled the Grid heterogeneity problem by developing security assurance and risk-resilient strategies and offer eight job-scheduling algorithms for use under various risky conditions. We design three risk-resilient scheduling strategies, namely, *preemptive, replication-based*, and *delay-tolerant*. We then incorporate them into existing heuristics and genetic algorithms. Furthermore, we observe that, while these heuristics or genetic algorithms are easy to implement with low complexity, security

- *S. Song is with the Internet and Grid Research Laboratory, University of Southern California, 3740 McClintock Ave., EEB 110, Los Angeles, CA 90089-2562. E-mail: shanshan.song@alumni.usc.edu.*
- *K. Hwang is with the Department of EE-Systems, University of Southern California, 3740 McClintock Ave., EEB 212, Los Angeles, CA 90089-2562. E-mail: kaihwang@usc.edu.*
- *Y.-K. Kwok is with the Department of Electrical and Electronic Engineering, Chow Yei Ching Bldg., University of Hong Kong, Pokfulam Road, Hong Kong, China. E-mail: ykwok@hku.hk.*

Manuscript received 22 June 2005; revised 17 Feb. 2006; accepted 27 Feb. 2006; published online 21 Apr. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0207-0605.

0018-9340/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society



Grid Job Submission

Computing Power: 120 Tflops

Deadline: Mon, 28 Feb 2005, 00:00:00

Budget: $100,000

Security Demand: High

(a) (b)

Fig. 1. The Grid job outsourcing environment and security demand in job submission. (a) Grid job-scheduling scenario. (b) Grid job submission interface.

assurance and, thus, better performance could be achieved, if we build the risk-resistant features into Grid job scheduling algorithms.

In this regard, we propose a new fast GA that has two salient features: 1) Its search is history-sensitive and, thus, converges much faster than a traditional GA and 2) its encoding is based on the messy concept, which is shown to be effective in protecting good building blocks in the chromosomes [27]. Moreover, unlike a traditional GA, the messy encoding scheme can inherently handle the three resilient scheduling strategies. In summary, this paper has made basic fundamental contributions in three aspects:

1. Design and evaluation of three Grid job-scheduling strategies, which can be applied to enable security assurance in existing heuristic or genetic algorithms. These risk-resilient scheduling algorithms are experimentally proven effective in real-life implementations on Grids or on any distributed computing systems.
2. Design and evaluation of a new *space-time genetic algorithm* (STGA) based on the messy encoding concept with a space-time guided search mechanism. Risk-resilient strategies are applied to provide security assurance for trusted Grid computing.
3. After extensive performance study of eight job scheduling algorithms, we reveal their relative strengths, weaknesses, and applicability in scalable Grid computing. In particular, our findings suggest that it is more resilient to tolerating job delays by calculated risky conditioning, instead of resorting to job preemption, replication, or assuming risky operations in Grid computing systems.

The performance of the heuristic algorithms and the STGA are evaluated with two practical workloads: NAS (*Numerical Aerodynamic Simulation*) and PSA (*Parameter-Sweep Application*). We use a set of performance metrics for Grids, namely, the *makespan, average turnaround time, Grid utilization, slowdown ratio, replication overhead*, and *failure rate*. We propose to use a five-dimensional Kiviat graph to assess the quality of Grid services (QoGS) in Section 6.2.

The rest of the paper is organized as follows: Section 2 presents a brief review of related work and identifies our unique approach to solving the scheduling problem over risky Grid platforms. In Section 3, we present a job failure model and specify the design strategies applied to heuristic scheduling driven by security demand. In Section 4, we describe our new STGA scheme. We present performance metrics for Grids and report extensive simulation results in Section 5. We reveal the relative performance and scalability of the eight scheduling algorithms with Kiviat graphs in Section 6. Finally, we summarize the contributions and make some remarks on further research in Section 7.

2 Related Work and Our New Approach

We first review related work on trusted and security-aware resource allocation in computational Grid. Then, we introduce our unique approach to solving the security problem in Grid job scheduling.

2.1 Related Previous Work

As pointed out by many researchers [3], [17], [30], trust and security are two different notions. *Security* is a notion associated with the assurance of secure computing services by a Grid site or by a cluster node, whereas trust is reflected by the behavior of a resource node. These two terms are correlated by many attributes to be discussed in Section 2.2. In our approach, we consider both the *security demand* of user jobs and the *trust index* of resource nodes. We focus on how to evaluate the trust index of machine nodes and how they affect the successful execution of user jobs. We use a defensive and corrective approach—preventing or avoiding security-triggered failures from happening or damaging user jobs, if trust assurance could be established to match with user demands at the job scheduling time.

In the past, job scheduling has been primarily suggested for supercomputers, real-time, and parallel computers [22], [26], [29], [33]. In security-aware job scheduling, the scheduling process becomes much more challenging [19], [40], [42]. Unfortunately, well-known scheduling approaches for Grid computing largely ignore this security factor, with only a handful of exceptions. Most notably, Azzedin and Maheswaran [3] suggested integrating the trust concept into Grid resource management. They proposed a trust model that incorporates the security implications into scheduling algorithms. In our approach, we focus on how the risk brought the security concerns and how they affect the overall performance of the jobs in the system.

Humphrey and Thompson [19] provided usage models for security-aware Grid computing. However, they did not elaborate on how a scheduler should be designed to address the security concerns in collaborative computing over distributed cluster environment. Abawajy [1] suggested a distributed scheme to provide fault tolerance for job execution in a Grid environment. This scheme replicates jobs at multiple sites to guarantee successful job executions. Another related previous research effort is by Dogan and Ozguner [13], [14].

Hwang and Kesselman [23] pointed out that the Grid environment is inherently unreliable by nature. They provided a failure detection service and a flexible failure-handling framework as a fault-tolerant mechanism on the Grid. They tackle the problem by providing a remedial method—remedy jobs when failure is observed. In [42], [43], Xie et al. studied security-aware scheduling for cluster and Grid applications. Their studies address the applications demand of both real-time performance and security.

We briefly review below the study of resource allocation and job scheduling in computational Grid. Berman et al. developed adaptive application level scheduling (AppLeS) for Grid computing [7]. In AppLeS, each application is assigned with a scheduling agent to monitor available resources and generate job schedule. Buyya et al. proposed a deadline and budget constrained scheduling model [9]. They used a computational economy framework for regulating the supply and demand for resources and allocating them for applications based on the users quality of services requirements. In et al. proposed a framework for policy constraint enforcement strategies for Grid job computing [25]. Wu and Sun [41] developed a *Grid Harvest Service* (GHS) scheduler to monitor long-running application progress and detect possible resource abnormality. Other researchers study dynamic [16], QoS-driven [18], coallocation [11], and decoupled [12] scheduling methods of Grid job scheduling.

Our work is also built upon research that tackles trust management. In [44], Xiong and Liu suggested a P2P reputation system called *PeerTrust*, which maintains a composite trust value for each peer. Other researchers have proposed methods for the propagation and management of trust and distrust. Lin et al. [30] assumed that trust condition could be derived from Grid security enhancement. The distinction in our approach lies in an optimized matching of security requirements and supports judicious Grid site mapping for user jobs. This obviously transcends the maintenance of reputation values to provide feedback to Grid sites.

There have been several recent advancements in tackling the replication-based job scheduling problem. Bansal et al. [4] suggested minimizing the turnaround time of a parallel application using replication. Their scheme works by judiciously examining the opening holes in an existing unreplicated schedule, followed by filling the holes with critical jobs. Bajaj and Agrawal [5] recently reported a replication approach to minimize the makespan of a parallel application in a heterogeneous distributed system. However, the TANH algorithm is designed only for precedence-constrained jobs and is thus unsuitable for large-scale parallel applications.

2.2 Our New Approach

Our work is built upon the related work on Grid security, trust management, and job scheduling. Our approach matches trust requirements by user jobs with a judicious security index at Grid sites, which extends security-aware Grid job scheduling in the direction of delay tolerance and job replications. We reveal the trade-offs between speed performance and security assurance. We look deeper into *replication-based* algorithms and *delay-tolerant* strategies, which were largely ignored in the past. This motivates us to design new heuristics and genetic algorithms for optimized Grid job scheduling under security constraints.

Grid sites may exhibit unacceptable security conditions and system vulnerabilities [19]. Worry about inevitable security threats and doubtful trustworthiness of remote resources has created barriers in trusted job outsourcing to remote computer platforms. In mapping jobs onto Grid sites, we tackle a completely new dimension of security-related problems. The first step is for a user job to issue a *security demand* (*SD*) to all available resource sites. The trust model requires assessing the resource site's trustworthiness, called the *trust level* (*TL*) of a node.

The TL quantifies how much a user can trust a site for successfully executing a given job. A job is expected to be successfully carried out when *SD* and *TL* satisfy a *security-assurance condition* ($SD \leq TL$) during the job mapping process. The process of matching *TL* with *SD* is similar to the real-life scenario where the Yahoo! portal requires users to specify the security level of the login session. On a realistic platform, both *SD* and *TL* are highly loaded concepts, as depicted in Fig. 2.

Our definition of trust extends from the one defined in [3], where trust is only associated with the behavior of a resource site. In this paper, *trust level* is an aggregation of both *behavior attributes* and intrinsic *security attributes*. The *trust level* is an aggregation of five major attributes listed in Fig. 2a. They are behavior attributes accumulated from the historical performance of a resource site, such as *prior job execution success rate* and *cumulative site utilization*, and intrinsic security attributes such as *intrusion detection*, *firewall*, and *intrusion response* capabilities.

The intrinsic attributes can be measured as *intrusion detection rate, false alarm rate*, and *intrusion response results* [20], Both behavior and intrinsic attributes change dynamically and they depend heavily on the *security policy* [36], *accumulated reputation, self-defense capability, attack history, and site vulnerability*, etc. On the other hand, the users may be concerned in Fig. 2b about the *job sensitivity, peer authentication*, etc., in setting up their security demand.

In our earlier work [21], [37], [39], we have proposed a *fuzzy-logic-based trust model* to enable the aggregation of numerous trust parameters and security attributes into easy-to-use scalar quantities. Specifically, the *TL* is aggregated

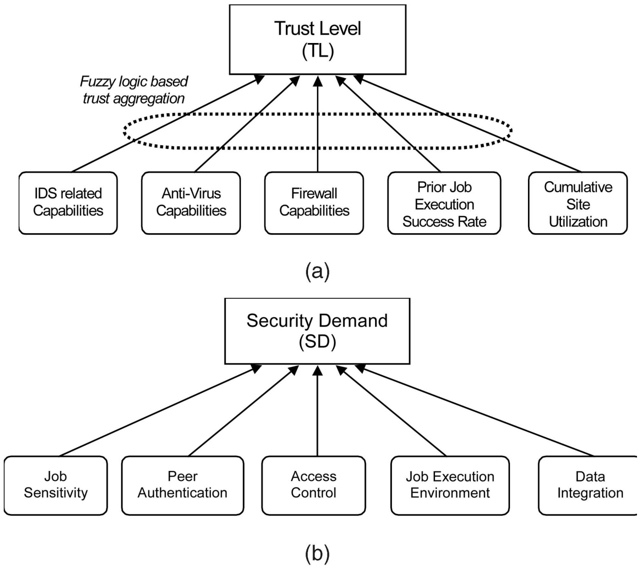


Fig. 2. Attributes affecting the trust level (*TL*) at the resource site and the security demand (*SD*) at the user end. (a) Site trust level from trust integration reported in [37]. (b) Job security demand.

through our fuzzy-logic inference process over the contributing parameters. A salient feature of our trust model is that, if a site's trust level cannot match with job security demand, i.e., $SD > TL$, our trust model could deduce detailed security features to guide the site security upgrade as a result of tuning the fuzzy system.

We propose three risk-resilient strategies, namely, *preemptive, replication*, and *delay-tolerant* strategies. The purpose is to reduce the risk involved in job scheduling. We derive risk-resilient job-scheduling algorithms, which are specially tailored for risky Grid environment. In this paper, we evaluate the performance of the min-min scheduling heuristic and genetic algorithm to illustrate the main concept of security binding. The security-driven algorithm design technique hereby developed can be applied to modify many other heuristics or genetic algorithms, such as the max-min, sufferage, or greedy etc. [8], [10].

3 Risk-Resilient Job Scheduling

When a remote site is infected by intrusions or by malicious attacks, that site may not be accessible by outside jobs or by the global job scheduler. The scheduler has two options to consider: 1) Abort the job scheduling and reschedule it later. 2) Allow the job's finish time to be delayed until the security barricade is removed. Three risk-tolerant strategies and eight scheduling algorithms are specified below.

3.1 Job Failure and Delayed Execution Models for Online Job Scheduling

In our model, a job could be delayed or dropped, if the site *TL* is lower than the job *SD*. The *SD* is a real fraction in the range $[0, 1]$ with 0 representing the lowest and 1 the highest security requirement. The *TL* is in the same range with 0 for the most risky resource site and 1 for a risk-free or fully trusted site. Specifically, we define the following job failure model as a function of the difference $SD - TL$ between the job demand and site trust:

1. **Job Failure Model.** The *job failure probability* is modeled by an exponential distribution given in (1). The failure coefficient λ is a fraction number. The negative exponent indicates failure grows with the difference $SD - TL$. The job abortion at a site could result from severe network attack or inaccessibility from a security imposed barricade.

$$P(complete\ fail) = \begin{cases} 0 & \text{if } SD \leq TL \\ 1 - e^{-\lambda(SD-TL)} & \text{if } SD > TL. \end{cases} \quad (1)$$

2. **Delayed Execution Model.** The probability of a delayed job execution is given in (2). The parameter γ decides the delayed execution against the difference caused by temporary resource unavailability. It is more likely that a job will be delayed versus a job failing. Thus, the delay coefficient γ is less than the failure coefficient λ.

$$P(delay) = \begin{cases} 0 & \text{if } SD \leq TL \\ 1 - e^{-\gamma(SD-TL)} & \text{if } SD > TL. \end{cases} \quad (2)$$

We consider a batch mode scheduling as modeled in Fig. 3. In our model, we assume a *fail-stop* execution [35], meaning that if a job fails on a site, then it will be rescheduled to restart at another site. Modeling the real-life scheduling situation, jobs are scheduled in batches. We assume that all jobs are scheduled independently. The main purpose of online job scheduling is to minimize the *total execution time* of all N jobs, which is often called the *makespan*, defined by:

$$\begin{aligned} &Makespan = \\ &Max_i\{FT(J_i) | FT(Ji) \text{ is the } finish\ time \text{ of job } J_i\}. \end{aligned} \quad (3)$$

3.2 Improved Heuristic and Genetic Scheduling Algorithms

Four online job scheduling strategies are specified below: These strategies can be applied to any existing heuristics and genetic algorithms to yield new scheduling algorithms. The preemptive, replication, and delay-tolerant strategies are all risk-resilient. Only the risky mode ignores the risk factors as assumed in conventional heuristics and genetic algorithms.

1. ***Risky mode.*** Jobs are scheduled independent of the risky or failing conditions of the resource site, meaning taking all the risks that may exist.
2. ***Preemptive mode.*** The scheduler preempts a job when a site is not accessible by the global scheduler, the job is assumed to be failing to meet the schedule requirement, and, thus, migrates to another Grid site.
3. ***Replicated mode.*** To play it safe, the scheduler duplicates the job to be executed at multiple sites, albeit all of them having $SD > TL$. All replicas will stop execution once one of the replicated jobs is successfully carried out.
4. ***Delay-Tolerant mode.*** The scheduler waits a period of time $\tau_i = \alpha T_i$ for job i, where T_i is the estimated execution time. The *delay factor* α indicates the percentage of time that can be tolerated by the global scheduler, i.e., the job must be finished within the

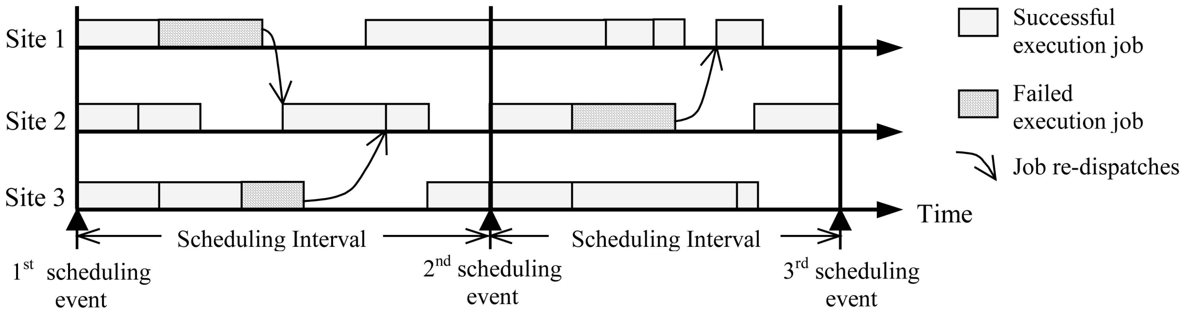


Fig. 3. System model for batch mode job scheduling in a computational Grid.

TABLE 1
Eight Grid Job Scheduling Algorithms Being Evaluated

Scheduling Strategy	Heuristic Algorithms	Genetic Algorithms
Risky job execution	**Risky-Heuristic:** Jobs are scheduled based on a heuristic algorithm without considering the risk factor.	**Risky-STGA:** Jobs are scheduled based on space-time genetic algorithm without considering the risk factor.
Preemptive insecure jobs	**P-Heuristic:** Preempts job execution due to insecure conditions. Resubmit failed jobs to other sites.	**P-STGA:** Job is scheduled based on STGA that allows preemption. Resubmit failed jobs to other sites.
Replicated job execution	**R-Heuristic:** Replicates jobs to multiple sites to prevent from possible failures (fixed number of replicas).	**R-STGA:** STGA that replicates jobs to multiple sites to prevent from possible failures (various number of replicas).
Delay-Tolerant scheduling	**DT-Heuristic:** Allows jobs to be delayed for some time when condition is not met before rescheduling.	**DT-STGA:** STGA that allows job to be delayed for some time, when condition is not met before rescheduling

extended time $\tau_i + T_i$. Beyond the tolerable period, the scheduler reschedules the job to other sites.

Eight security-aware job scheduling algorithms are introduced in Table 1 under the four strategies defined above. In this paper, we only apply the risk-resilient strategies on the min-min algorithm [8] and on a new STGA (*space-time genetic algorithm*) to be presented in Section 4. The applicable algorithms are not restricted to a particular heuristics applied. The *max-min, min-min, sufferage, greedy*, and genetic algorithms can all apply the proposed strategies [24]. The word "heuristic" in Table 1 can be replaced by any of the cited heuristic algorithms.

4 Space-Time Genetic Algorithm

In this section, we describe our proposed *Space Time Genetic Algorithm* (STGA) for risk-resilient scheduling of many jobs simultaneously over a large number of Grid sites.

4.1 A Traditional Risk-Taking Genetic Algorithm

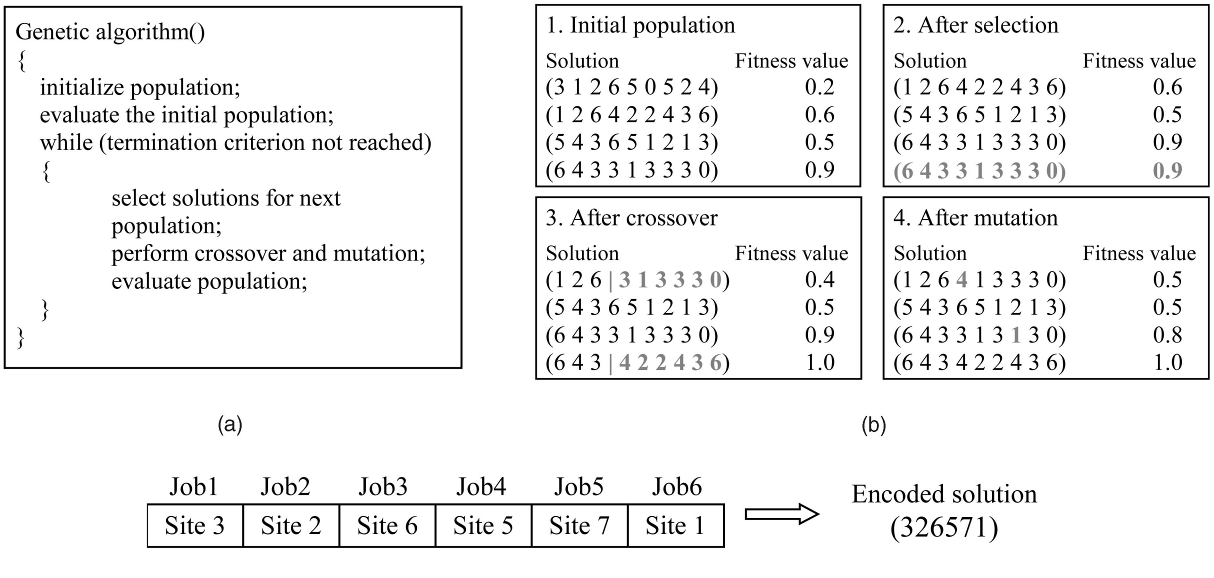
A *genetic algorithm* (GA) is a global search technique, which maintains a pool of potential solutions, called chromosomes [45]. The GA produces new solutions through randomly combining the good features of existing solutions. This exploratory searching step is achieved by using a *crossover* operator, which works by randomly exchanging portions of two chromosomes [8]. Fig. 4a shows the structure of a traditional GA, while Fig. 4b illustrates its operations, and a chromosome encoding scheme is given in Fig. 4c.

Crossover operation globally searches through the solution space. Another important local search operator is *mutation*, which works by randomly changing one of the genes in a chromosome. Mutation operation leads the search to get out of a local optimum. There is a *selection* process to remove the poor solutions. A value-based roulette wheel schema is used for selection. This schema probabilistically generates new population. Elitism, the property of guaranteeing the best solution to remain in the population, is also implemented.

The *crossover* and *mutation* operators are governed by their respective probabilities. The whole process is repeated a number of times, called *generations* or *iterations*. Here, each chromosome array is indexed with the site assignment for a job. The crossover operator causes random swapping of two portions of chromosomes. Note that the crossover point is randomly chosen. A mutation randomly changes the site assignment of a randomly selected job in an arbitrary chromosome. The crossover operator is the major facility to explore the search space to locate good solutions.

4.2 A New Encoding Scheme for Risk-Resilient Scheduling

Typically, a GA is composed of two main components, which are problem dependent: the *evaluation function* and the *encoding schema*. The evaluation function measures the quality of a particular solution. Each solution is associated with a fitness value, which is represented by the completion time of the schedule. In this case, the smallest fitness value represents the best solution. The efficiency of a GA depends on the encoding scheme applied. The typical chromosome encoding shown in Fig. 3c suffers from a deficiency—it cannot handle



```
Genetic algorithm()
{
   initialize population;
   evaluate the initial population;
   while (termination criterion not reached)
   {
         select solutions for next
         population;
         perform crossover and mutation;
         evaluate population;
   }
}
```

(a)

1. Initial population

Solution	Fitness value
(3 1 2 6 5 0 5 2 4)	0.2
(1 2 6 4 2 2 4 3 6)	0.6
(5 4 3 6 5 1 2 1 3)	0.5
(6 4 3 3 1 3 3 3 0)	0.9

2. After selection

Solution	Fitness value
(1 2 6 4 2 2 4 3 6)	0.6
(5 4 3 6 5 1 2 1 3)	0.5
(6 4 3 3 1 3 3 3 0)	0.9
(6 4 3 3 1 3 3 3 0)	0.9

3. After crossover

Solution	Fitness value
(1 2 6 | 3 1 3 3 3 0)	0.4
(5 4 3 6 5 1 2 1 3)	0.5
(6 4 3 3 1 3 3 3 0)	0.9
(6 4 3 | 4 2 2 4 3 6)	1.0

4. After mutation

Solution	Fitness value
(1 2 6 4 1 3 3 3 0)	0.5
(5 4 3 6 5 1 2 1 3)	0.5
(6 4 3 3 1 3 1 3 0)	0.8
(6 4 3 4 2 2 4 3 6)	1.0

(b)

Job1	Job2	Job3	Job4	Job5	Job6
Site 3	Site 2	Site 6	Site 5	Site 7	Site 1

(c)

Fig. 4. Structure and operations of a traditional genetic algorithm. (a) Structure of a typical GA. (b) Major operations in a GA. (c) A typical chromosome encoding of a possible solution.

security assured scheduling. Thus, we propose using the *messy representation* [27] as defined in Fig. 5.

Here, a chromosome is a list of ordered pairs (*job ID, site ID*). For example, (3, 4) represents the third job being assigned to the fourth Grid site. The length of each chromosome can be different. Furthermore, a gene's value may be *overspecified*, i.e., it may appear more than once in a chromosome with different values. In the scheduling context, each gene represents a (job, site) pair. The *underspecified* genes do not exist in the chromosome. We use a prespecified *template* to assign their values. The template is the best solution found in previous iterations. A randomly generated template is used for the first iteration.

In Fig. 5, *Job* 3 and *Job* 5 are not mapped to any site. By applying the predefined template, they are scheduled to *Site* 2 and *Site* 3, respectively. In our simulation, the template is the best solution found so far. We use the *cut and splice* operator shown in Fig. 6. Two chromosomes are randomly selected and a random cut point is chosen on each of them. Depending on the cut probability chosen, the two chromosomes are cut randomly. The resulting pieces are exchanged and spliced together. These operators protect the good building blocks existing in a chromosome [27].

We use the *cut and splice* operator shown in Fig. 6 to replace the crossover operation used for fixed length encoding. Two chromosomes are randomly selected and a random cut point is chosen on each of them. Depending on the cut probability chosen, the two chromosomes are cut randomly. The resulting pieces are exchanged and spliced together. These operators protect the good building blocks existing in a chromosome [27]. The solution generated from the *cut and splice* operation may be underspecified. Again, the template assigns values for underspecified genes, which guarantees that solutions generated by the cut-and-splice operation are valid.

4.3 Risk-Resilient Genetic Algorithms

After a number of rounds of evolutions, the genetic algorithm generates a final solution. Based on the messy encoding schema, we design three risk-resilient genetic algorithms:

1. **P-STGA.** Preemptive genetic algorithms scan the final solution from left to right on a first come first serve basis. A job is first scheduled to the site identified by the leftmost (job, site) pair. The remaining sites are backups. If a failure is observed in this site, the job will be migrated to the first backup site. If more failure is observed, the job will be migrated to the second backup site until the job is successfully executed or exhausts all backup sites. In the worst case, the job will be resubmitted as a new

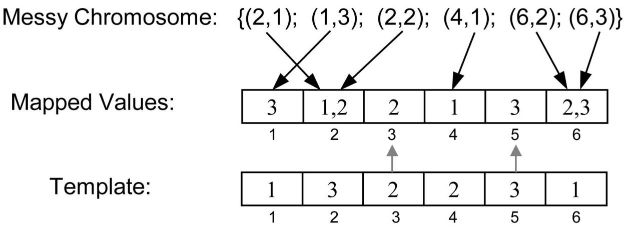

Messy Chromosome: {(2,1); (1,3); (2,2); (4,1); (6,2); (6,3)}
Mapped Values:
3
1,2
2
1
3
2,3
1
2
3
4
5
6
Template:
1
3
2
2
3
1
1
2
3
4
5
6

Fig. 5. A messy-encoded chromosome for introducing redundancy needed for security-assurance and risk resilience.

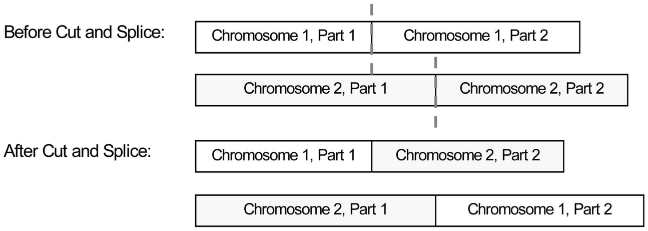


Fig. 6. Cut and splice operations over the messy-encoded chromosomes.

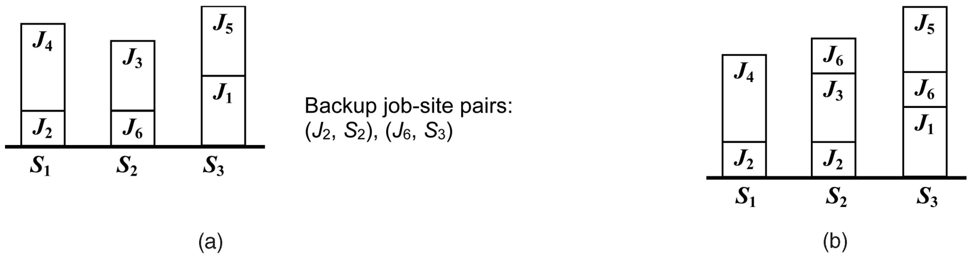
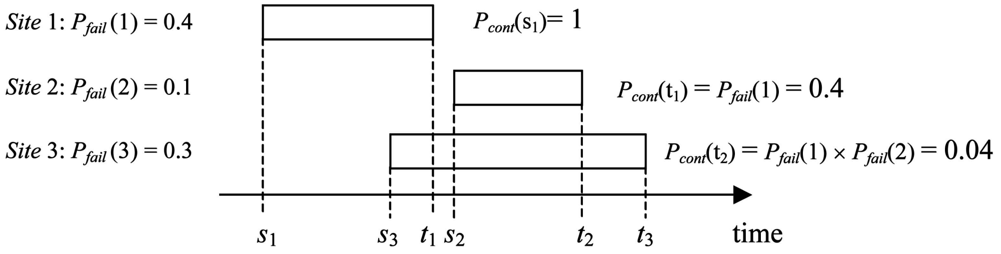


Fig. 7. Scheduling results based on the messy-encoded chromosome in Fig. 5. (a) Preemptive scheduling. (b) Replication scheduling.


Site 1: $P_{fail}(1) = 0.4$
$P_{cont}(s_1) = 1$
Site 2: $P_{fail}(2) = 0.1$
$P_{cont}(t_1) = P_{fail}(1) = 0.4$
Site 3: $P_{fail}(3) = 0.3$
$P_{cont}(t_2) = P_{fail}(1) \times P_{fail}(2) = 0.04$
s_1
s_3
t_1
s_2
t_2
t_3
time

Site 1: $E(T_1) = P_{cont}(s_1) \times (t_1 - s_1) = t_1 - s_1$

Site 2: $E(T_2) = P_{cont}(s_2) \times (t_2 - s_2) = 0.4 \times (t_2 - s_2)$

Site 3: $E(T_3) = P_{cont}(s_3) \times (t_1 - s_3) + P_{cont}(t_1) \times (t_2 - t_1) + P_{cont}(t_2) \times (t_3 - t_2) = t_1 - s_3 + 0.4 \times (t_2 - t_1) + 0.04 \times (t_3 - t_2)$

Fig. 8. Replicated genetic algorithm calculates the expected job execution time based on job failure probability.

job in the next batch. We use a simple example to illustrate the initial scheduling result of P-STGA in Fig. 7a, where *Job* 2 is scheduled to *Site* 1 and *Site* 2 acting as two backup sites.

2. **R-STGA.** A replicated genetic scheme schedules a job to selected sites. Thus, a job could be scheduled to multiple sites, albeit all of them satisfying $SD > TL$. Once the job is finished at one site, all other sites are informed to stop the job execution to save computing power. If the job fails at all sites, it will be resubmitted as a new job in the next batch. The scheduling result of R-STGA is illustrated in Fig. 7b.
3. **DT-STGA.** The initial scheduling result of DT-STGA is the same as P-STGA. A job is only scheduled to the leftmost encoded site. The difference is that, once a job failure is observed, the DT-STGA allows the site to have a period of time for job recovery. If the job cannot be recovered, it will be migrated to a backup site.

The fixed-encoded STGA schedules jobs with risks. Thus, we named the algorithm Risky-STGA, which implies that they are not risk resilient.

To evaluate the fitness of a messy-encoded chromosome, we design a probability-based evaluation function. This new evaluation function estimates the expected execution time of a job on a site based on the job failure probability. We use a simple example in Fig. 8 to explain the basic concept involved. Fig. 8 shows the snapshot of an example R-STGA job scheduling. A job is dispatched to three sites for replicated execution. Suppose that the job failure probabilities on three sites are $P_{fail}(1) = 0.4$, $P_{fail}(2) = 0.1$, and $P_{fail}(3) = 0.3$. If the job starts to execute independently at time s_1, s_2, and s_3, and if the job is successfully executed, then the job will be finished at time t_1, t_2, and t_3, respectively.

In R-STGA, if a job is successfully executed at one site, all replicas at other sites will stop execution. For example, in Fig. 8, if the replica on *Site* 1 succeeds, then the replica on *Site* 2 will not start and the replica on *Site* 3 will stop at time t_1. Therefore, the probability that the job will continue after time t_1 will be $P_{cont}(t_1) = P_{fail}(1)$. In general, in a Grid of m sites, for a specific site l, the job continues after time t_l only if all previous executed replicas fail. Thus, the probability that the job will continue after time t_l will be:

$$P_{cont}(t_l) = \prod_{\substack{1 \le i \le m \\ t_i \le t_l}} P_{fail}(i). \tag{4}$$

As illustrated in Fig. 8, the execution time of each replica is broken into multiple pieces by t_i $(1 \le i \le m)$. Each piece has an execution probability. The expected execution time $E(T_l)$ of a replica on an l is calculated by (5), which is applicable to both P-STGA and DT-STGA.

$$E(T_l) = P_{cont}(s_l) \times (t_s - s_l) + \sum_{\substack{1 \le i,j \le m \\ s_l \le t_i < t_j \le t_l \\ \neg \exists t_k, t_i < t_k < t_j}} P_{cont}(t_i) \times (t_j - t_i), \tag{5}$$

where

$$P_{cont}(s_l) = \begin{cases} P_{cont}(s_1) & l = 1 \\ P_{cont}(t_i) & l \ne 1 \end{cases}$$

and $s_l \ge t_i$ and $\neg \exists t_j, s_l > t_j > t_i$.

The calculated expected times are used to evaluate the fitness of messy encoded chromosomes.

4.4 Acceleration of the Genetic Search Process

The crux of the *high efficiency* in our STGA approach is illustrated in Fig. 9. In general, starting from a set of randomly generated initial pool of solutions (called the initial chromosome population), the quality of the best

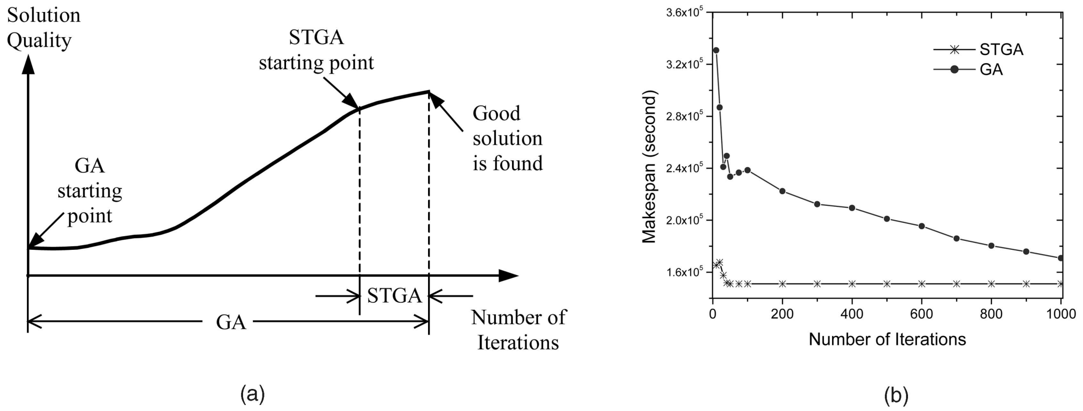
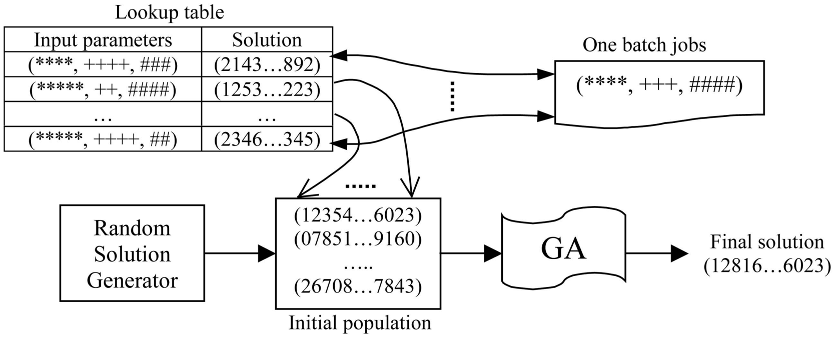


Fig. 9. The STGA starts with a high-quality initial population and converges to a good solution in 50 evolutional iterations, as compared with thousands of iterations needed for a conventional GA starting with a low-quality population. (a) GA versus STGA in terms of iterations. (b) The STGA converges in 50 iterations.



Input parameters	Solution
(****, ++++, ###)	(2143…892)
(*****, ++, ####)	(1253…223)
…	…
(*****, ++++, ##)	(2346…345)

Fig. 10. Space-time genetic algorithm (STGA) for trusted Grid job scheduling.

solution in the pool is usually quite low. Over a number of generations, the solution quality gradually improves. Thus, the question is how to skip the initial phase of struggling with bad solutions. Our answer to this question is to make use of *prior scheduling knowledge*—the scheduling results from previous executions.

The justification of this space-time approach is that the workload submitted to a practical Grid computing platform usually has some time correlation or temporal locality. That is, the jobs submitted previously would appear again in the near future. For instance, a physicist trying to generate some simulation data today would very likely try them again tomorrow. Thus, in our STGA, we keep a history table storing the job specifications and their schedules. The historical data on scheduling are used to form the initial population when the GA is invoked. This results in an elevated starting point of the STGA algorithm, as shown in Fig. 9a.

Fig. 9b shows the Grid PSA benchmark results on the number of evolutional iterations needed to yield a good solution with low makespan time for both the GA and STGA algorithms. The STGA converges after only 50 iterations to a very good solution. The conventional GA takes thousands of iterations to reach a comparable solution. This clearly demonstrates the advantage of STGA over the GA in large-scale Grid computations.

The overall scheduling mechanism of our proposed STGA is illustrated in Fig. 10. In the lookup table, input parameters are represented by 3-tuples, corresponding to: 1) the next available times of Grid sites, 2) job execution time matrix, and 3) job security demands. The 3-tuples of new input jobs are compared with those in the table entries. The solutions of those matched 3-tuples are included in the initial populations.

We map each parameter into a vector and then we use the vector comparison method to calculate the similarity for each parameter. Suppose two vectors are $\boldsymbol{a} = (a_1, a_2, , \dots, a_k)$ and $\boldsymbol{b} = (b_1, b_2, \dots, b_k)$, the similarity between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is defined by:

$$Similarity(a, b) = 1 - \frac{\sum_{i=1}^{k} \|a_i - b_i\|}{\max(a_i, b_i, i = 1, 2, \dots k)}. \qquad (6)$$

Because the initial population is based on a prior solution, we expect higher solution quality in the STGA scheme. Our scheme not only performs evolutionary searching over the solution space as manifested by the chromosome population, but also spans over the time period represented by scheduling history. In simulation experiments, we use the Min-Min heuristics to schedule a small number of training jobs to generate the initial lookup table. The *least recently matched* entries are replaced by new job scheduling solutions.

5 Simulated Performance Results on NAS and PSA Workloads

In this section, we define several performance metrics for measuring Grid performance. Then, we present the

TABLE 2
Simulation Parameter Settings in NAS and PSA Experiments

Parameter	**Value setting**
Number of jobs ***N***	NAS: 16000; PSA: 10000
Number of sites ***M***	NAS: 12; PSA: 20
Job arrival rate	NAS: Given by trace; PSA: 0.008jobs/second/site
Job workloads	NAS: Given by trace; PSA: 20 levels (0 – 300000)
Site processing speed	NAS: 8×8 nodes and 4×16 nodes; PSA: 10 levels (0 – 10)
Job security demands (*SD*)	0.6 - 0.9, uniform distribution
Site trust level (*TL*)	0.3 - 1.0, uniform distribution
Failure and delay coefficients	$\lambda = 3$; $\gamma = 2$
Delay factor α	$\alpha = 0.2$

simulated benchmark results on scheduling a large number of user jobs from the commonly tested NAS and PSA benchmark suites in Grid systems.

5.1 NAS and PSA Workloads and Performance Metrics

In order to gain practical insights into the effectiveness of the scheduling approaches, we use two realistic workloads to test the scheduling effectiveness in a risky Grid environment.

The NAS Workload. We use the *Numerical Aerodynamic Simulation* (NAS) benchmark recorded on the 128-node iPSC/860 system at the NASA Ames Research Center. This NAS trace contains 92 days of data gathered in 1993. We simulate 12 Grid sites with eight sites containing eight nodes each and the remaining four sites containing 16 nodes each. The NAS traces key parameters for simulated jobs —the *arrival time, job size* [32]. These parameters could be applied to both the homogenous and heterogeneous environments. This trace was sanitized to remove the user specified information and preprocessed to correct for system downtime. Table 2 summarizes key simulation parameters used in the NAS benchmark experiments [15].

Parameter-Sweep Application (PSA) Workload. The *parameter-sweep application* (PSA) model has emerged as a killer application benchmark for high-throughput computing applications in large-scale Grid experiments [10]. The PSA was designed to model N independent jobs (each with the same task specification over a different data sets) on M distributed sites, where N is much greater than M. In our simulation, we assume that the input to each job is a set of files and each job only produces a single output file.

The PSA benchmark is ideal to model N independent jobs running in the SPMD mode with the same task specification over different data sets. In Table 2, we have specified the PSA workload with 10,000 jobs running on 20 simulated Grid sites. For both the NAS and PSA workloads, the user SD is set uniformly distributed between 0.3 and 0.9. The site TL is set uniformly distributed between 0.3 and 1.0. The failure rate and delay factor are also set with the same values for all fairness in comparative studies.

Performance Metric Evaluated. To evaluate the scheduling performance, we propose using the following performance metrics:

- ***Average turnaround time.*** Let N be the total number of jobs, c_i, be the completion time for a single job J_i, and a_i the job arrival time. The *average turnaround time* is defined by:

$$Average\ job\ turnaround\ time = \sum_{i=1}^{N} (c_i - a_i)/N. \quad (7)$$

- ***Makespan.*** This was defined as $\max\{c_i, i = 1, 2, ..N\}$ in (3).
- ***Slowdown ratio.*** Denote the start time for a single job J_i as b_i, the average waiting time is $\sum_{i=1}^{N} (c_i - b_i)/N$. The slowdown ratio is the ratio between the average turnaround time and the average waiting time. This metric indicates the contention experienced in a security-aware scheduling algorithms.

$$\begin{aligned} Slowdown\ \ ratio &= \frac{\sum_{i=1}^{N}(c_i - a_i)/N}{\sum_{i=1}^{N}(c_i - b_i)/N} \\ &= \frac{\sum_{i=1}^{N}(c_i - a_i)}{\sum_{i=1}^{N}(c_i - b_i)}. \end{aligned} \quad (8)$$

- ***Job failure rate.*** When $TL < SD$, job execution may fail due to insecure resource sites applied. The numerator, N_{fail}, is the number of failed and rescheduled jobs. The failure rate is thus defined by the ratio:

$$Job\ failure\ rate = N_{fail}/N. \quad (9)$$

- ***Grid utilization.*** Defined by the percentage of processing power allocated to successfully executed jobs out of the total processing power available of a global Grid. For replication strategy, only one replica is counted as a successfully executed job. The Grid utilization does include processing resources consumed by stopped jobs.
- ***Replication overhead ratio (ROR).*** In replicated job scheduling, let $R_{replica}$ be the resources allocated to replicated jobs and R_{total} be the total resources allocated to all jobs. ROR measures the percentage of extra resources needed to execute the job replicas.

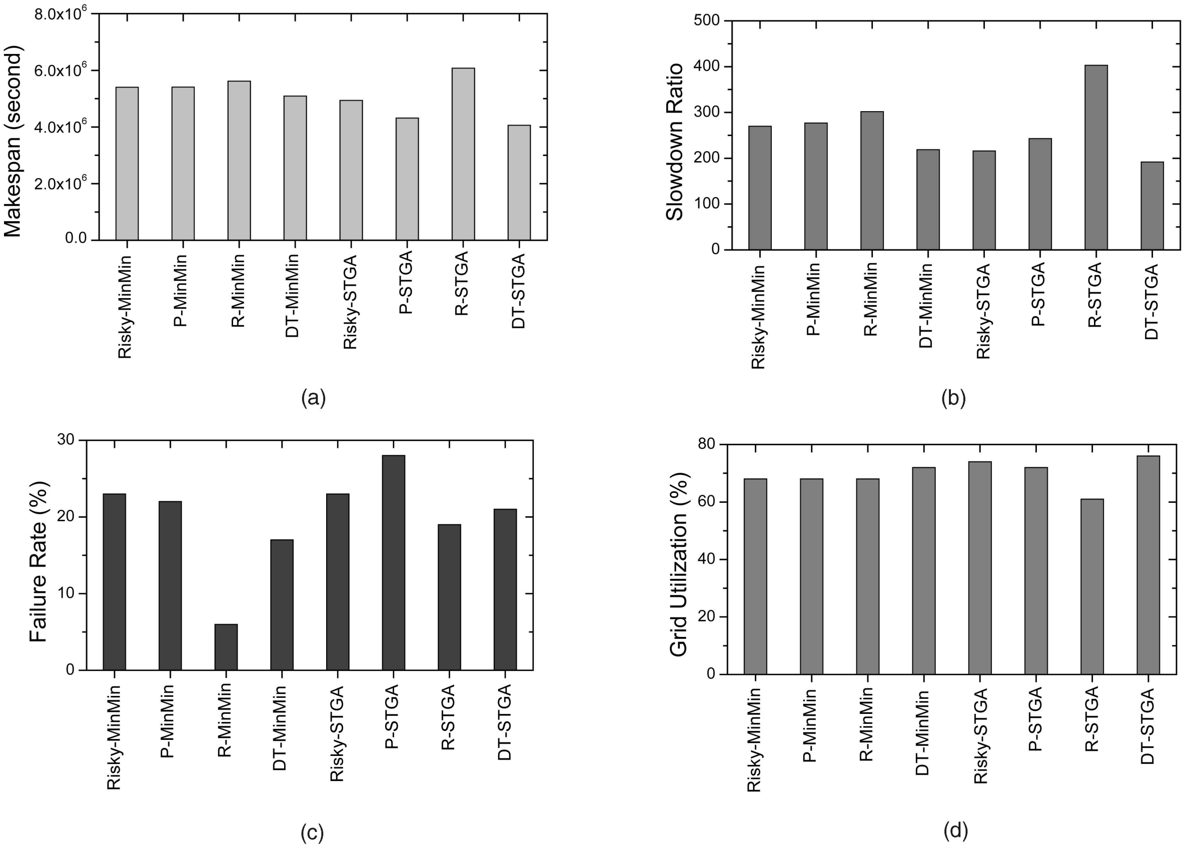


Fig. 11. Performance results of using security-assured heuristics and risk-resilient STGA algorithms to schedule 16,000 NAS jobs over 12 simulated Grid sites. (a) Makespan. (b) Slowdown ratio. (c) Failure rate. (d) Grid utilization.

$$ROR = R_{replica}/R_{total}. \tag{10}$$

5.2 Performance Results over the NAS Workload

We evaluated the performance of eight variants of the Min-Min heuristic and space-time genetic algorithms with the NAS trace workload. The simulation results are given in Fig. 11 for each metric proposed. Fig. 11a shows the *makespan* results of eight algorithms. Overall, the P-STGA and DT-STGA algorithms have the lowest makespan performance. The R-STGA performs the worst. All others have essentially the same level of performance. For both the Min-Min and STGA algorithms, the *delay-tolerant* strategy performs better than others and the *replication* strategy performs the worst.

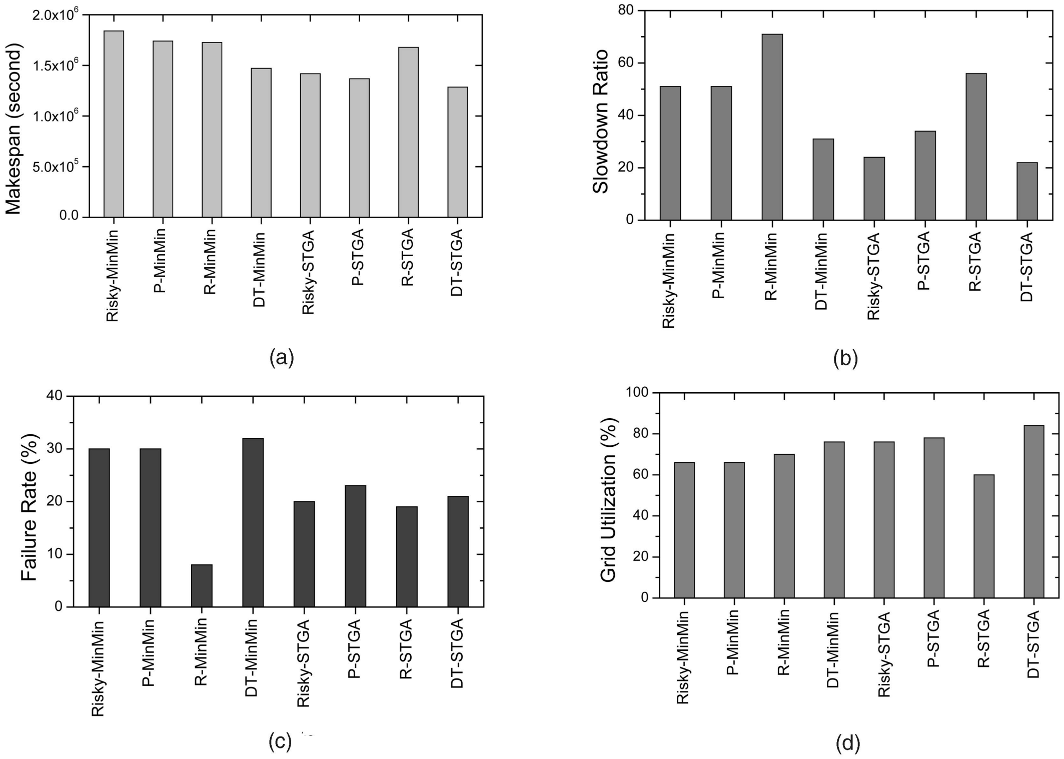
The superior makespan performance of the *delay-tolerant* algorithms is due to their resilient allocation of jobs to the machines. Specifically, the overall makespan is not affected much if the job failures do not trigger a collapse of jobs across machines. On the other hand, the *replication* approach is simply too pessimistic in that it tends to avoid failures completely, instead of proactively handling them. As a result, heavy overheads were incurred to prolong the makespan.

Fig. 11b shows the *slowdown ratio* results. A large slowdown ratio is observed from the R-STGA algorithm. A small slowdown ratio is observed on the DT-STGA and DT-Min-Min algorithms. The superior makespan performance of the *delay-tolerant* algorithms is due to their resilient allocation of jobs to the machines. Specifically, the overall makespan is not affected much if the job failures do not trigger a collapse of jobs across machines. On the other hand, the *replication* approach is simply too pessimistic in that it tends to avoid failures completely, instead of proactively handling them.

Fig. 11c shows the *job failure rates* of eight algorithms. Overall, Min-Min heuristic algorithms have lower failure rates than the space-time genetic algorithms. In particular, the R-MinMin algorithm has very few job failures, only 6 percent. Furthermore, among the three risky strategies, *replication algorithms* always have the lowest failure rate, the *delay-tolerant policies* have the medium failure rate, and the *preemptive* algorithms have the highest failure rate. Fig. 11d shows that the DT-STGA algorithm has the highest utilization and the R-STGA has the lowest utilization.

All other algorithms have comparable utilization rates. Indeed, a rather uniform utilization rate across all algorithms indicates that failure-avoidance approaches (e.g., *replication*) cannot necessarily save computational resources. The rationale is that, in using the *delay-tolerant* algorithm, the utilization rate can be kept at an acceptable level, while enhancing the overall performance. In summary, the turnaround time results are highly correlated with the makespan results. Thus, the conservative approaches (e.g., *replication*) tend to have longer turnaround times due to extra computations performed. As to failure rate, a replicated scheduling approach may expect a lower failure rate due to redundancy applied.

However, the *delay-tolerant* algorithm exhibits resilient behaviors with a moderate level of failures, which does not necessarily lead to a worse performance. No single algorithm achieves the highest performance for all metrics.



(a)

(b)

(c)

(d)

Fig. 12. Performance results of using security-assured min-min heuristics and risk-resilient STGA algorithms to schedule 10,000 PSA jobs over 20 simulated Grid sites. (a) Makespan. (b) Slowdown ratio. (c) Failure rate. (d) Grid utilization.

The *delay-tolerant risky* algorithm achieves the best in makespan, turnaround time, slowdown ratio, and Grid utilization. However, its failure rate is relatively higher compared with the replicated strategy. The *replication strategy* has the lowest failure rate; however, it has the worst performance in all other metrics.

5.3 Performance Results over the PSA Workload

In Fig. 12, we evaluate the performance of the Min-Min heuristic and genetic algorithms under the PSA workload. We simulated the execution of a PSA workload of 10,000 jobs over 20 Grid sites. Fig. 12a shows the makespan results of all algorithms. First, compared with the Min-Min algorithms, the STGA algorithms exhibit shorter makespan. Second, among all eight algorithms, the Min-Min heuristics algorithm exhibits the longest makespan. Third, comparing all three strategies, the *delay-tolerant strategy* has the shortest makespan. The *replication strategy* has the longest makespan.

The PSA workload is irregularly structured, with massive parallelism at the job level. This is reflected by the fact that the *delay-tolerant* algorithm is more robust and resilient to changes in computational patterns. In Fig. 12b, large slowdown ratios are observed from two replication strategies. Small slowdown ratios are observed from the DT-Min-Min, STGA, and DT-STGA algorithms. Fig. 12c shows the *job failure rates*. Overall, the R-Min-Min algorithm has very few job failures, roughly 8 percent.

Among the three strategies, *replication algorithms* always have the lowest failure rate, *delay-tolerant algorithms* have the medium failure rate, and *preemptive* algorithms have the highest failure rate. In Fig. 12d, the DT-STGA algorithm has the highest utilization and the R-STGA has the lowest utilization rate. The NAS workload demands more computing power than the PSA workload. This translates to longer turnaround time and larger slowdown ratios.

The makespan and Grid utilization have relatively the same performance ranking order under both workloads. The R-MinMin algorithm under the PSA workload has a much higher slowdown ratio and turnaround time than those for the NAS workload. The NAS workload is more balanced in terms of site computing power distribution. The PSA workload does not enjoy such a nice distribution. Under both workloads, the R-Min-Min algorithm has the lowest failure rate. This is due to the fact that every job is duplicated in both workloads. The higher failure rate of the DT-Min-Min algorithm under the PSA workload is attributed to its job irregularity.

5.4 Replication Overhead Ratio and Effect of Waiting Period

Replicated job execution on different sites is a redundancy technique to tolerate site failure in a Grid. As seen in previous sections, the Grid performance in terms of the makespan of all replicated jobs may not pay off. This is due to the fact that replicated job execution demands extra resources. Thus, there exists a trade-off between the costs of enhanced security resilience and extra resources used. It is useful to consider this trade-off to work out an appropriate operating range in using the replicated job scheduling policy.

The defense heterogeneity induces some unacceptable job schedules by which the waiting is excessively long or replication is too costly. A practical solution is to aggregate the job requirements as well as the site capabilities to scalar quantities *TL* and *SD*. Even if the aggregated trustworthiness of a resource site is sufficiently high, jobs dispatched to such sites can still fail. Consequently, the failed jobs need to

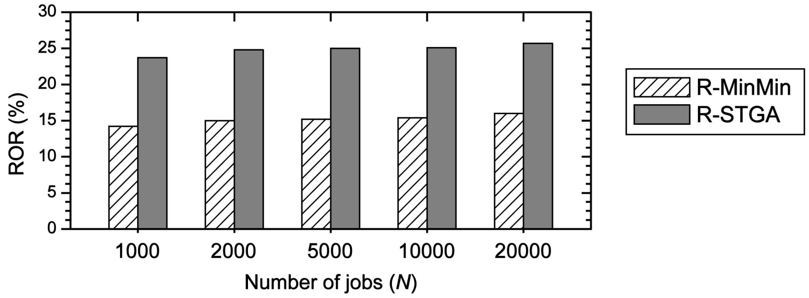


Fig. 13. Replication overhead ratios of two replicated job scheduling algorithms under the scalable PSA workload.

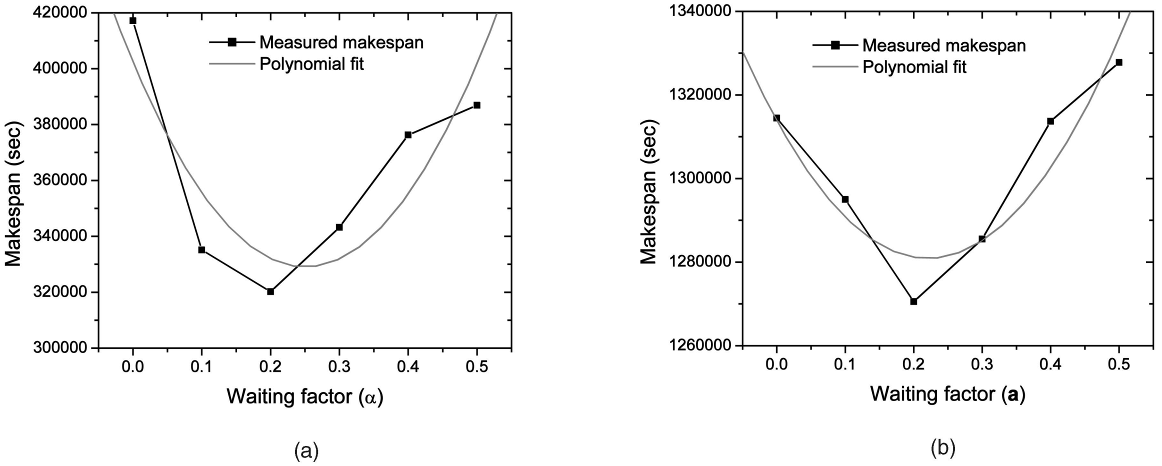


Fig. 14. Effect of the waiting factor α on the makespan of two delay-tolerant job scheduling algorithms under the PSA workload. (a) DT-MinMin ($N = 2,000$). (b) DT-STGA ($N = 5,000$).

be rescheduled. Here, we study the overhead of job replication of two algorithms: the R-Min-Min and R-STGA.

Based on simulation results under the scalable PSA workload, the ROR of the R-MinMin and R-STGA algorithms is plotted in Fig. 13 against the job sizes. Here, we assumed double redundancy by which every risky job is duplicated one copy for parallel execution. The overhead ratio measured is essentially flat, independent of the job sizes. On the average, the R-STGA algorithm uses 25 percent extra resources for the replicated jobs. There is 15 percent extra resources for replicated job execution in using the R-Min-Min algorithm. The R-STGA algorithm uses more extra resources than even some jobs satisfying SD < TL. This situation is not allowed in the R-Min-Min algorithm. This also explains the lower Grid utilization rate in using the R-STGA algorithm.

The selection of a sufficiently small waiting period (α) is crucial to the ultimate performance a delay-tolerant algorithm. The waiting period could be sensitive to the application structures and, hence, may need to be dynamically adjusted to avoid waste of resources. To illustrate the effects of using different waiting periods, we performed additional experiments by varying the waiting factor α. Fig. 14 plots the makespan results of using the DT-MinMin and DT-STGA algorithms in scheduling 2,000 and 5,000 jobs, respectively, under the PSA workload.

The results indicate that neither a small nor a large delay leads to a better performance. In other words, the delay extension cannot be made too short or too long. Indeed, if the global scheduler allows a short delay, only a small amount of jobs can benefit from the extension, thus the performance improvement is not obvious. However, if the global scheduler allows a long delay, many jobs will take longer times than expected, thus degrading the global performance. Fig. 14 plots the effects of delayed job execution on total makespan performance.

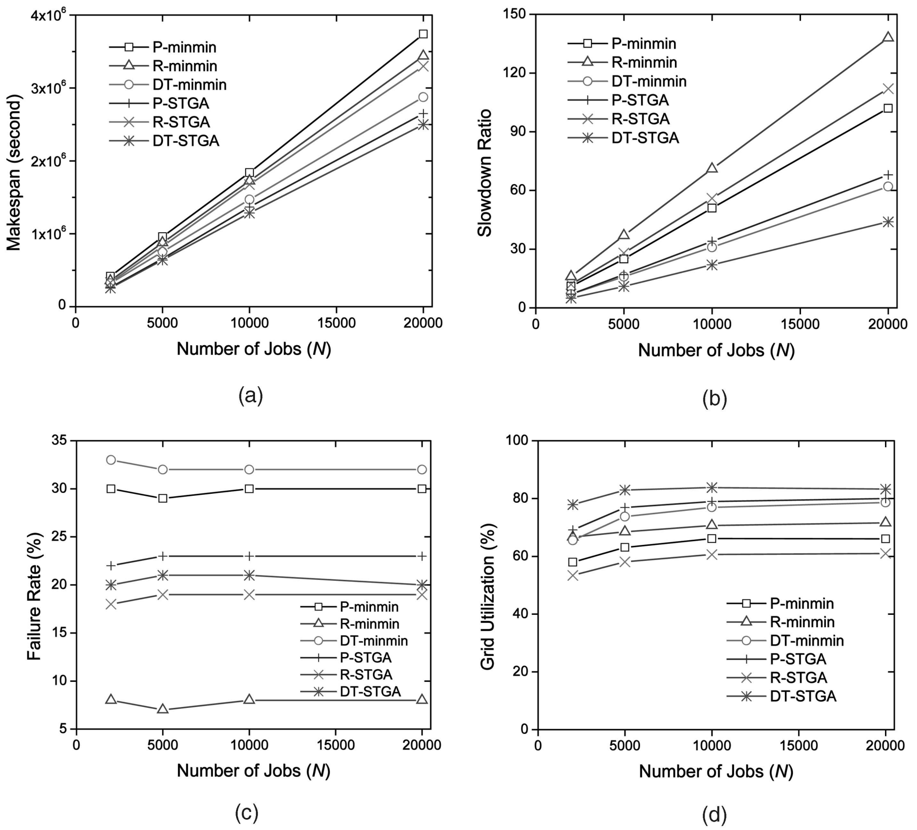
In practice, we find the optimal choice of $\alpha = 0.2$ at the bottom of the plotted curves. A polynomial fitting-function is used to calculate the optimal α. It is interesting to see $\alpha = 0.2$ is optimal for both delay-tolerant algorithms. This is likely due to the compensation for the 20 percent delayed time with the overhead incurred with implementing the delay-tolerant algorithms. The above results reported in Fig. 13 and Fig. 14 are also a good indication of the low overhead incurred in both the replicated and delay-tolerant security-aware Grid job scheduling algorithms.

6 Relative Performance and Scalability Analysis

In this section, we discuss the scaling effects and aggregate performance of eight job-scheduling algorithms specified in Table 1, considering their combined strength and weakness in five performance dimensions.

6.1 Scaling Effects of Workload Size (Number of Jobs)

The more user jobs injected into a Grid system, the higher the workload and the longer the time needed to process the submitted jobs. The performance effects of varying the



(a)

(b)

(c)

(d)

Fig. 15. Scaling effects of the number of jobs on the performance of six security-assured scheduling algorithms in executing scalable PSA workload over 20 Grid resource sites. (a) Makespan. (b) Slowdown ratio. (c) Failure rate. (d) Grid utilization.

number of simulated jobs are reported in Fig. 15. Because the number of jobs in the NAS trace workload is fixed at 20,000, we study the scaling effects only on the PSA workload. The PSA workload scales from $N = 2,000$ to 20,000 jobs in our study over 20 simulated Grid sites.

All simulation parameters follow the setting in Table 2. With more jobs running the same PSA workload, the *makespan* and the *slowdown ratio* increase with the same pace, as seen in Fig. 15a and Fig. 15b. The DT-STGA has the best performance for its capability to tolerate some time delays with suffering too much in makespan. Furthermore, the flat job failure rate in Fig. 15c shows the robustness of these algorithms. The slow increase of Grid site utilization in Fig. 15d indicates that all algorithms reach their maximum utilization by job scaling.

The higher slope of the Makespan curve or of the slowdown ratio shows the lower scalability of a given algorithm. The flat curves in Fig. 15c and Fig. 15d demonstrate that the job failure rate and site utilization are almost independent of the problem sizes. This is due to the fact that PSA has all independent jobs. The R-Min-Min algorithm shows the lowest failure rate and the DT-STGA has the highest site utilization. This is a common situation for all replication algorithms to have a low failure rate due to redundancy introduced. The delay-tolerant algorithms have higher utilization because jobs are delayed if they are in danger of failing due to mismatch between the SD and TL.

6.2 Kiviat Graph for Measuring Grid Service Quality

We use the five-dimensional Kiviat graph to measure the aggregate Grid performance. Each Kiviat graph has five orthogonal dimensions, each representing one of the performance metrics defined in Section 5.2. The scales and ranges of five measures are specified in Fig. 16a. The circle center represents zero for all measures. The range of *makespan, turnaround time*, and *slowdown ratio* is from zero to the largest value observed in the experiments. The *failure rate* and *underutilization rate* (1-*utilization*) are within the interval $[0, 1]$. Fig. 16 contains eight Kiviat graphs, each of which shows the simulated NAS benchmark performance of a Grid job scheduling algorithms being evaluated.

These Kiviat graphs provide a holistic view of the capabilities of each algorithm. Based on visualizing these graphs, we can select the best approach under practical constraints in real-life applications. Specifically, the smaller the shaded area, A_{shaded}, on the center of the Kiviat graph, the better the *Quality of Grid Services* (QoGS) provided by a scheduling algorithm. The entire circle area, A_{circle}, provides a common denominator to compare the relative performance. Thus, we define below the QoGS of a scheduling algorithm. This compound performance metric measures the overall quality on the scheduling algorithm being evaluated.

$$\text{QoGS} = \left(1 - \frac{A_{shaded}}{A_{circle}}\right) \times 100\%. \tag{11}$$

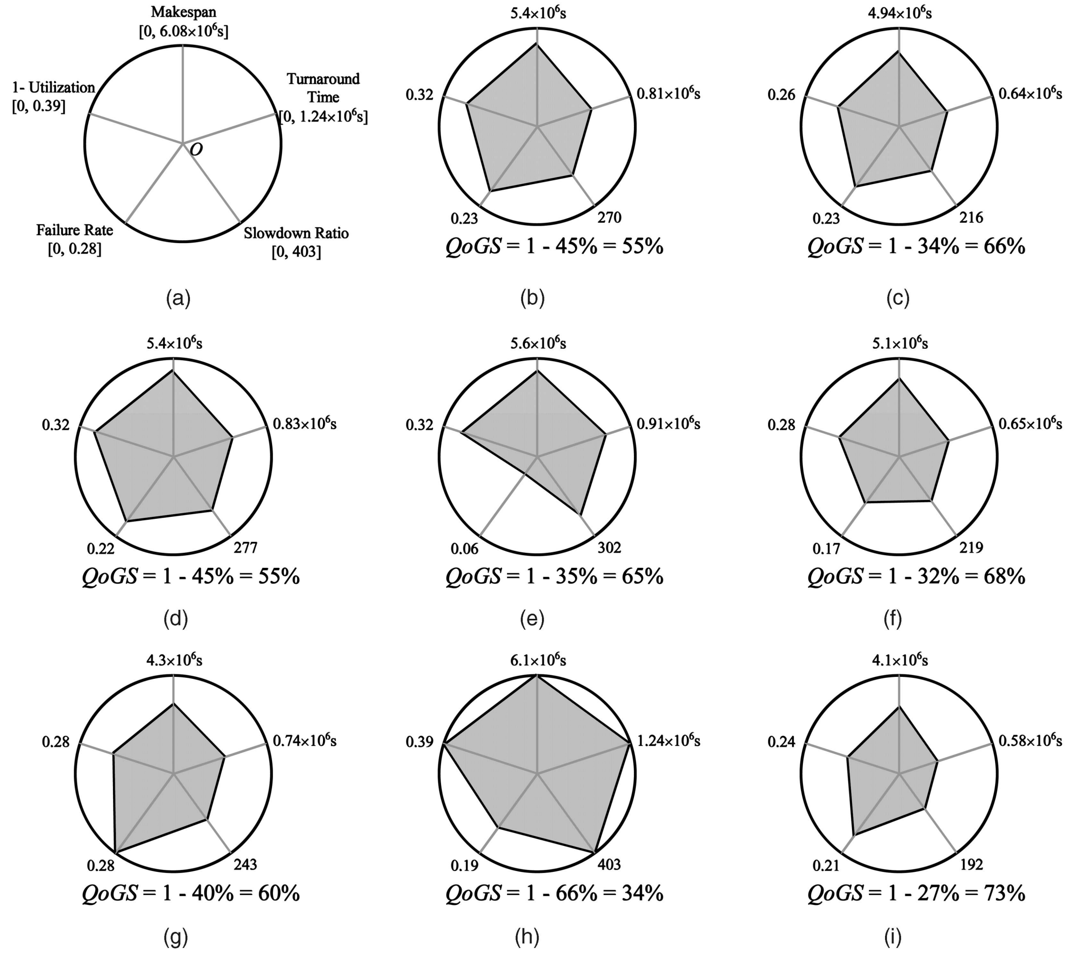


Fig. 16. Kiviat graphs showing the aggregate performance of six security-assured job scheduling algorithms (d) through (i), compared with two risky scheduling algorithms in (a) and (b) under the Grid NAS benchmark workload. (a) Scales and ranges of five measures. (b) Risky MinMin. (c) Risky STGA. (d) P-MinMin. (e) R-MinMin. (f) DT-MinMin. (g) P-STGA. (h) R-STGA. (i) DT-STGA.

The DT-STGA and DT-Min-Min algorithms achieve the highest QoGS. This results from the fact that the delayed job execution can assure better security to gain in all five dimensions. The R-STGA has the lowest QoGS, as shown by the largest area in Fig. 16h. The two replicated algorithms suffer in all five dimensions. Based on these results, we conclude that the delay-tolerant strategy is indeed the most resilient job-scheduling algorithm. This algorithm performs well at all performance dimensions, to yield the highest QoGS.

From a practical point of view, our approach is more suitable for use in a dynamic heterogeneous Grid or peer-to-peer environment because Grid sites join and leave the system at will and unexpectedly. In summary, the Kiviat diagrams reveal the overall Grid performance on a given benchmark suite. We demonstrate here the aggregate performance results for NAS only. The NAS results reflect practical computational Grid performance.

6.3 Ranking of Security-Assured Job Scheduling Algorithms

We summarize in Table 3 the QoGS and the relative ranking of eight scheduling algorithms for both the NAS and PSA workloads. The DT-SGTA ranks first for both workloads. The DT-Min-Min and STGA demonstrate high performance quality under both workloads. The preemptive and replication algorithms rank low among all six security-aware algorithms. The implication here is that the delay-tolerant strategy should be adapted for general-purpose Grid job scheduling. The replicated algorithms are used only if the job failure is the primary concern.

The performance of both replicated algorithms is inferior in all dimensions. One plausible explanation is that the replicated strategies in an open Grid environment are not guided by structures known to the scheduler. Specifically, there is no critical path for the replication algorithms to follow through in order to select the most critical jobs to replicate with respect to the application structure and the network structure. Consequently, many replicated jobs are simply a waste of extra computational resources allocated. However, the replication overhead is rather low, less than 25 percent in using the STGA scheme and less than 15 percent in using the R-Min-Min algorithm under the PSA workload, as reported in Fig. 13.

The entries in Table 3 lead to several important conclusions:

TABLE 3
Quality of Grid Services and Ranking of Eight Grid Job Scheduling Algorithms

Scheduling Algorithms	NAS workload		PSA workload	
	QoGS	Ranking	QoGS	Ranking
Risky Min-Min	55%	6th	47%	8th
Preemptive Min-Min	55%	6th	48%	7th
Replicated Min-Min	65%	4th	53%	6th
Delay-tolerant Min-Min	68%	2nd	70%	4th
Risky STGA	66%	3rd	79%	2nd
Preemptive STGA	60%	5th	77%	3rd
Replicated STGA	34%	7th	58%	5th
Delay-tolerant STGA	73%	1st	84%	1st

1. The dynamic STGA algorithms perform better than the static Min-Min heuristics.
2. The DT algorithms appeal better to computational Grids.
3. The performance of the replicated and preemptive algorithms is barely tolerable.
4. The risky heuristics should not be applied at all in Grid computing for their poor performance and QoGS.

However, the risky STGA and P-STGA are not bad at all. The relative ranking of algorithms is subject to change with different workloads applied.

7 Conclusions and Implementation Requirements

In a Grid computing environment, when a resource site is under attack, the site may be unresponsive but not in a total down state. The jobs dispatched to such a failing site may take higher risk and potentially may not succeed. There are also situations where a site is shut down and all jobs must be purged. The real challenge to successful Grid scheduling is thus to decide what the smart alternatives are or to have contingent plans under different levels of risky conditions. The classical results in fault-tolerant computing cannot be applied here on Grids due to large-scale heterogeneity encountered in Grid resource distribution [34].

7.1 Summary of Research Findings

In summary, we have considered two categories of trust: 1) intersite trust, for which a fundamental level of support exist, and 2) job-site trust, for which a scheduling algorithm is needed to do the security matching. The NAS and PSA benchmark results prove that the security-aware job scheduling algorithms scale well with the increase of both Grid size and the workload (number of jobs). Scalability and robustness are the two major assets built into the proposed security-aware Grid job scheduling algorithms. The low overhead in extra resources and time delays are also useful incentives to insist on security assurance.

Summarized below are the major contributions and research findings out of our NAS and PSA benchmark experiments in a simulated Grid environment.

1. First, we developed the job failure model and the delayed execution model for job scheduling under security threats in Grids. These models characterize various levels of risk involved in Grid job scheduling. These conditions are applied to bind security in upgrading heuristics or STGA algorithms for large-scale Grid job scheduling.
2. Second, we propose three risk-resilient strategies: the preemptive, replicated, and delay-tolerant strategies for Grid job scheduling. Based on these scheduling strategies, we proposed six security-assured job scheduling algorithms to enable risk-resilient services under different risky conditions. These strategies cover various Grid applications, not necessarily restricted to the NAS or PSA workloads being reported.
3. The relative performances of the six security-assured scheduling algorithms are evaluated with the NAS and PSA results. We measure the makespan, average job turnaround time, Grid utilization, slowdown ratio, job failure rate, and replication overhead of these algorithms. We find that the scheduling resource overheads of both replicated and delay-tolerant algorithms are kept rather low (15 to 25 percent at most). The Kiviat graph is newly proposed for use to reveal the quality of aggregate Grid computing services provided,
4. Our simulation results suggest that, in a wide-area Grid environment, it is more resilient for the global job scheduler to tolerate the job delays, instead of resorting to preemption or replication or assuming unrealistic risk-free models. The risky scheduling heuristics have the lowest performance for being unable to adapt to changes in detected risky conditions.
5. The delay-tolerant (DT) heuristic or STGA algorithms have the best performance among the eight algorithms evaluated. The rationale is to use available computing power more intelligently. The DT algorithms preserve the resource utilization rate. These algorithms exhibit resilient capability with the least failure rate. To sum up, all risk-resilient scheduling algorithms scale well with the increase in job numbers.

7.2 Implementation Requirements and Limitations

How efficient implementation of the security-assured scheduling algorithms is depends on the trust model used in a real-life Grid environment. We considered the cases that the participating Grid sites are periodically assessed with three trust conditions: 1) trusted communication of SD and TL values, 2) trusted execution of replicated jobs, and 3) trusted propagation of job results after various delayed executions. Condition 1) implies that communication links among Grid sites are secure. This is achievable in practice because Grid sites can set up secure IP tunnels among them. As to conditions 2) and 3), execution of replicated jobs and propagation of delayed job results can be implemented by having the Grid sites digitally sign the data and code (e.g., SD, TL, job results) so that the recipients can verify the data integrity and authenticate all parties involved.

As the Grid sites are administered by different organizations, the defense capabilities of the sites may differ dramatically. Such a heterogeneity feature is highly complex to prevent a perfect match of the job security requirements with the trust indices of various resource sites located in different administrative domains. In [37], we have proposed a fuzzy-logic-based trust integration and security binding methodology to align security requirements of both jobs and Grid resource sites. Detailed implementation requirements of the fuzzy trust management system can be found in the companion papers [37], [38].

Furthermore, such a fundamental trust index is not in conflict with the concept that a job may not be executed at a certain site if the SD and TL values are not matched. Indeed, a low TL value (with respect to the SD of a job) does not mean that the site is totally not trustworthy. Rather, it only means that the site's security capabilities are not sufficiently high to assure a successful execution of a particular set of jobs. For examples, the job may require contacting some external clients (outside the Grid), which may create even more security breaches. If the site executing this job does not have a strong firewall, the job may fail, etc.

Of course, we are limited by the assumption of performing cooperative work among all participating Grid sites. The limitation is posed by exchange of trust index information and agreement to enhance the trust level, when the peer evaluation scores are low. In this sense, our scheme may not be applicable to selfish Grids or noncooperative Grid nodes. Game theory can be applied to alleviate the limitation imposed by selfish Grids [28], which is beyond the scope of this paper. It would be interesting to combine the two approaches in further research work.

Acknowledgments

The funding support of this work by US National Science Foundation ITR Grant ACI-0325409 is appreciated. The authors appreciate the critical comments from the anonymous referees, which have inspired them to add new results on replication overhead and slowdown analysis in Section 5.4 and discuss the implementation and limitation issues in Section 7.2. They are grateful for useful discussions and suggestions made by their fellow team members at the University of Southern California Internet and Grid Research Laboratory.

References

[1] J.H. Abawajy, "Fault-Tolerant Scheduling Policy for Grid Computing Systems," *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '04),* Apr. 2004.

[2] M.J. Atallah, K.N. Pantazopoulos, J.R. Rice, and E.H. Spafford, "Secure Outsourcing of Scientific Computations," *Advances in Computers,* vol. 54, chapter 6, pp. 215-272, 2001.

[3] F. Azzedin and M. Maheswaran, "Integrating Trust into Grid Resource Management Systems," *Proc. Int'l Conf. Parallel Processing,* Aug. 2002.

[4] S. Bansal, P. Kumar, and K. Singh, "An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 6, pp. 533-544, June 2003.

[5] R. Bajaj and D.P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 2, pp. 107-118, Feb. 2004.

[6] *Grid Computing: Making the Global Infrastructure a Reality,* F. Berman, G. Fox, and T. Hey, eds. John Wiley & Sons, 2003.

[7] F. Berman et al., "Adaptive Computing on the Grid Using AppLeS," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 4, pp. 369-382, Apr. 2003.

[8] T.D. Braun, D. Hensgen, R. Freund, H.J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing,* vol. 61, no. 6, pp. 810-837, 2001.

[9] R. Buyya, M. Murshed, and D. Abramson, "A Deadline and Budge Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications,* 2002.

[10] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," *Proc. Heterogeneous Computing Workshop (HCW),* 2000.

[11] K. Czajkowski, I. Foster, and C. Kesselman, "Resource Co-Allocation in Computational Grids," *Proc. IEEE Int'l Symp. High Performance Distributed Computing (HPDC-8),* 1999.

[12] H. Dail, F. Berman, and H. Casanova, "A Decoupled Scheduling Approach for Grid Application Development Environments," *J. Parallel and Distributed Computing,* vol. 63, pp. 505-524, 2003.

[13] A. Dogan and F. Ozguner, "Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 308-323, Mar. 2002.

[14] A. Dogan and F. Ozguner, "A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. Int'l Conf. Parallel Processing (ICPP '02),* pp. 352-358, Aug. 2002.

[15] D.G. Feitelson and B. Nitzberg, "Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860," Research Report RC 19790 (87657), IBM T.J. Watson Research Center, Oct. 1994.

[16] N. Fujimoto and K. Hagihara, "Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid," *Proc. Int'l Conf. Parallel Processing,* 2003.

[17] M. Gupta, P. Judge, and M. Ammar, "A Reputation System for Peer-to-Peer Networks," *Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV),* 2003.

[18] X. He, X.H. Sun, and G. Laszewski, "A QoS Guided Scheduling Algorithm for the Computational Grid," *Proc. Workshop Grid and Cooperative Computing,* Dec. 2002.

[19] M. Humphrey and M.R. Thompson, "Security Implications of Typical Grid Computing Usage Scenarios," *Proc. High Performance Distributed Computing,* Aug. 2001.

[20] K. Hwang, Y. Chen, and H. Liu, "Protecting Network-Centric Computing System from Intrusive and Anomalous Attacks," *keynote paper, Proc. IEEE Workshop System and Network Security (SNS-225),* Apr. 2005.

[21] K. Hwang, Y.K. Kwok, S. Song, M. Cai, Y. Chen, and Y. Chen, "Security Binding and Worm/DDoS Defense Infrastructure for Trusted Grid Computing," *Int'l J. Critical Infrastructures,* 2005.

[22] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming.* San Francisco: McGraw-Hill, Feb. 1998.

[23] S. Hwang and C. Kesselman, "A Flexible Framework for Fault Tolerance in the Grid," *J. Grid Computing,* vol. 1, no. 3, pp. 251-272, 2003.

[24] O.H. Ibarra and C.E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *J. ACM,* vol. 24, no. 2, pp. 280-289, Apr. 1977.

[25] J. In, P. Avery, R. Cavanaygh, and S. Ranka, "Policy-Based Scheduling for Simple Quality of Service in Grid Computing," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2004),* Apr. 2004.

[26] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy of Grid Resource Management Systems for Distributed Computing," *Software-Practice and Experience,* vol. 32, no. 2, pp. 135-164, 2002.

[27] D. Knjazew and D.E. Goldberg, "Solving Permutation Problems with the Ordering Messy Genetic Algorithm," *Advances in Evolutionary Computing,* A. Ghosh and S. Tsutsui, eds., pp. 321-350, Springer, 2003.

[28] Y.K. Kwok, S. Song, and K. Hwang, "Selfish Grid Computing: Game-Theoretic Modeling and NAS Performance Results," *ACM/IEEE CC-Grid-2005,* May 2005.

[29] Y.K. Kwok and I. Ahmad, "Link-Constrained Scheduling and Mapping of Tasks and Messages to a Network of Heterogeneous Processors," *Cluster Computing,* vol. 3, no. 2, pp. 113-124, Sept. 2000.

[30] C. Lin, V. Varadharajan, Y. Wang, and V. Pruthi, "Enhancing Grid Security with Trust Management," *Proc. Services Computing 2004 (SCC 2004),* 2004.

[31] C. Liu, L. Yang, I. Foster, and D. Angulo, "Design and Evaluation of a Resource Selection Framework for Grid Applications," *Proc. Int'l Symp. High Performance Distributed Computing (HPDC-11),* 2002.

[32] V. Lo and J. Mache, "Job Scheduling for Prime Time vs. Non-Prime Time," *Proc. IEEE Cluster Computing,* 2002.

[33] M. Maheswaran, S. Ali, and H.J. Sigel, "Dynamic Mapping and Scheduling of Independent Tasks onto Heterogeneous Computing Systems," *J. Parallel and Distributed Computing,* pp. 107-131, 1999.

[34] X. Qin, H. Jiang, and D.R. Swanson, "An Efficient Fault-Tolerant Scheduling Algorithm for Real-Time Tasks with Precedence Constraints in Heterogeneous Systems," *Proc. Int'l Conf. Parallel Processing,* pp. 360-368, Aug. 2002.

[35] F.B. Schneider, "Byzantine Generals in Action: Implementing Failstop Processors," *ACM Trans. Computer System,* vol. 2, no. 2, pp. 145-154, 1984.

[36] F. Siebenlist, N. Nagaratnam, V. Welch, and C. Neuman, "Security for Virtual Organizations: Federating Trust and Policy Domains," *The Grid: Blueprint for a New Computing Infrastructure,* I. Foster and C. Kesselman, eds., pp. 353-387, Elsevier, 2004.

[37] S. Song, K. Hwang, R Zhou, and Y.K. Kwok, "Trusted P2P Transactions with Fuzzy Reputation Aggregation," *IEEE Internet Computing,* pp. 18-28, Nov/Dec. 2005.

[38] S. Song, K. Hwang, and Y.K. Kwok, "Trusted Grid Computing with Security Binding and Trust Integration," *J. Grid Computing,* Sept. 2005.

[39] S. Song, Y.K. Kwok, and K. Hwang, "Trusted Job Scheduling in Open Computational Grids: Security-Driven Heuristics and a Fast Genetic Algorithm," *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '05),* Apr. 2005.

[40] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for Grid Services," *Proc. Int'l Symp. High Performance Distributed Computing (HPDC-12),* 2003.

[41] M. Wu and X. Sun, "A General Self-Adaptive Task Scheduling System for Non-Dedicated Heterogeneous Computing," *Proc. IEEE Int'l Conf. Cluster Computing,* Dec. 2003.

[42] T. Xie and X. Qin, "Enahancing Security of Real-Time Applications on Grids through Dynamic Scheduling," *Proc. 11th Workshop Job Scheduling Strategies for Parallel Processing (JSSPP2005),* pp. 146-158, June 2005.

[43] T. Xie, X. Qin, and A. Sung, "SAREC: A Security-Aware Scheduling Strategy for Real-Time Application on Clusters," *Proc. Int'l Conf. Parallel Processing (ICPP-2005),* pp. 5-12, June 2005.

[44] L. Xiong and L. Liu, "PeerTrust: Supporting Reputation-Based Trust to P2P E-Communities," *IEEE Trans. Knowledge and Data Eng.,* pp. 843-857, July 2004.

[45] A.Y. Zomaya, R.C. Lee, and S. Olariu, "An Introduction to Genetic-Based Scheduling in Parallel-Processor Systems," *Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Science,* A.Y. Zomaya, F. Ercal, and S. Olariu, eds., pp. 111-133, chapter 5. New York: Wiley, 2001.

Shanshan Song received the BS degree in computer science from the University of Science and Technology of China in 2001 and the PhD degree in computer science from the University of Southern California in 2005. She specializes in P2P networks, network security, database systems, parallel and distributed computing, and knowledge management. Her current research activities cover the areas of trust management in Grid and P2P systems, security-driven scheduling algorithms, cooperative game strategies, and Grid computing systems. She is with the Internet and Grid Research Laboratory at the University of Southern California.

Kai Hwang received the PhD degree from the University of California, Berkeley, in 1972. He is a professor and director of the Internet and Grid Research Laboratory at the University of Southern California (USC). An IEEE fellow, he specializes in computer architecture, parallel processing, Internet and wireless security, Grid and cluster computing, and distributed computing systems. Dr. Hwang is the founding editor-in-chief of the *Journal of Parallel and Distributed Computing*. He is also on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems* and of the *International Journal of High-Performance Computing and Networking*. He has produced 17 PhD students at Purdue University and at USC. He has published more than 190 original scientific papers. He has written four textbooks and edited a number of research books including the series of *Annual Reviews in Scalable Computing* (World Scientific Publisher, 1999-2005). His latest two books, *Scalable Parallel Computing* (McGraw-Hill, 1998) and *Advanced Computer Architecture* (McGraw-Hill, 1993), are being adopted worldwide and translated into four languages. Presently, he leads a US National Science Foundation-supported GridSec project at USC, which develops security-binding techniques and distributed intrusion defense systems for trusted Grid or P2P computing.

Yu-Kwong Kwok received the PhD degree in computer science from the Hong Kong University of Science and Technology, Hong Kong, in 1997. He is an associate professor in the Department of Electrical and Electronic Engineering, University of Hong Kong. His research interests include Grid computing, mobile computing, wireless communications, network protocols, and distributed computing algorithms. He is on the editorial board of the *Journal of Parallel and Distributed Computing*. He is a member of the ACM, a senior member of the IEEE, and a member of the IEEE Computer Society and IEEE Communications Society.