Security-Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling

Shanshan Song, Yu-Kwong Kwok, and Kai Hwang*
University of Southern California, Los Angeles, CA 90089, USA
{*shanshas, yukwong, kaihwang*}*@usc.edu*

Abstract

In this paper, our contributions are two-fold: First, we enhance the Min-Min and Sufferage heuristics under three risk modes driven by security concerns. Second, we propose a new Space-Time Genetic Algorithm (STGA) for trusted job scheduling, which is very fast and easy to implement. Under our new model, a job can possibly fail if the site security level is lower than the job security demand. We consider three security-driven heuristic modes: secure, risky, and f-risky. The secure mode always dispatches jobs to secure sites meeting the job security demands. The risky mode allocates jobs to any available resource site, taking whatever the risk it may face. The f-risky mode tries to limit the risk to be at most certain probability f. Our extensive simulation results indicated that the proposed STGA is highly effective in scheduling two types of practical workloads: NAS (Numerical Aerodynamic Simulation) and PSA (parameter-sweep application). The STGA outperforms the Min-Min and Sufferage heuristics under three risk modes, in terms of a wide range of performance metrics including makespan, average response time, site utilization, slowdown ratio, and job failure rate.

Keywords: *Grid computing, on-line job scheduling, heterogeneous computing, security-driven heuristics, genetic algorithms, NAS benchmark, parameter-sweep applications, and distributed supercomputing.*

1. Introduction

Grid computing will eventually become commodities that are transported among different Grid sites (which can be large supercomputing centers or just small clusters) and executed by the sites in a transparent manner [4, 6]. Large-scale Grid computing demands a judicious job management system to address the security and privacy concerns [5, 9, 10, 14]. A job dispatched to a remote site may fail due to experiencing some infections or malicious attacks there. Specifically, a practical job scheduler must be *security-driven* in that it must consider the risk involved in dispatching jobs to remote sites. Unfortunately, existing Grid scheduling algorithms largely ignore this issue, making their applicability in a realistic environment rather doubtful.

Grids are most formed with resources owned by many organizations and thus are not dedicated for certain users. As such, jobs that are dispatched to a remote site can possibly experience some security and reliability problemsthe remote site may be intruded by some malicious users such that the jobs it is executing are destroyed. Simply put, such a situation can be modeled by a parameter called the *security level* (SL) that a Grid site can offer to remote jobs. Correspondingly, a job can be associated with a *security demand* (SD) value, so that if SD is not greater than SL, the job can expect to finish successfully; otherwise, the job may fail and has to be restarted on the same site or somewhere else [25, 26]. Here, as in nowadays real-life Grid computing usage scenarios, a job is an atomic unit of program execution that is neither malleable nor moldable.

It should be noted that this generic security model is used for illustration only and the parameters SL and SD could be composite structure (e.g., vectors) and their settings are largely administrative issues, which are beyond the scope of this paper [23, 24]. For instance, the value of SL can be dynamically managed by a local *intrusion detection system* (IDS) that continuously monitors the execution environment to see if there is any malicious code injected into the system. As such, SL and SD could also be a weighted

* This research was supported by an NSF ITR Research Grant under contract number ACI-0325409. Corresponding Author: Kai Hwang, Email: kaihwang@usc.edu, Tel: 213-740-4470, Fax: 213-740-4418. Y.-K. Kwok participated in this project while he was visiting USC, on his sabbatical leave from the University of Hong Kong (HKU). Kwok's research at USC was also supported by a research grant from the Research Grants Council of the HKSAR and the 2003-2004 Outstanding Young Researcher Award given by HKU.

sum of several system security parameters (e.g., job execution history, security levels of defense tools employed, etc.). Indeed, as detailed in [3, 24] by Azzedin and Maheswaran, SL and SD can be derived from many practical security or "trust" indices such as the "offered trust levels" and "requested trust levels". In our previous study [23], we have also designed a fuzzy logic based trust index that can also be used in our security-aware scheduling model considered in the current paper.

With a security-aware job execution model, job scheduling becomes more challenging. 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 observed that if jobs are dispatched solely based on the "trust-worthiness" of sites, then the security overheads can be very high. Thus, their research objective is to minimize those overheads. In our study, however, we focus on how the risk brought about by security concerns affects the overall performance of the jobs in the system in terms of slowdown ratio, makespan, site utilization, and failure rate.

There have been some previous research efforts related to our study. Humphrey and Thompson [16] provided a very useful discussion on various usage models for security-aware Grid computing. Abawajy [1] suggested a new scheduling approach called *Distributed Fault-Tolerant Scheduling* (DFTS) to provide fault-tolerance to job execution in a Grid environment. The DFTS algorithm works by replicating jobs at multiple sites in a careful manner so as to guarantee successful job executions. Another related previous research effort is by Dogan and Ozguner [12] who suggested a novel scheme for minimizing the failure rates of parallel applications on a dedicated heterogeneous computing platform.

In this paper, we first consider modifying the Min-Min and Sufferage scheduling heuristics to establish the secure scheduling framework [7]. These two heuristics are attractive choices for practical implementations, because they are fast and, as such, are suitable for online scheduling use in a real environment. To investigate their effectiveness in matching the SL of Grid sites to SD of jobs, we have developed three risk modes for each of the two heuristics:

- *Secure mode:* Allocate jobs only to those sites that can definitely satisfy the security requirements from the users. A job is allocated to an available site, only if the condition SD $\leq$ SL is met. The secure mode is considered as a conservative way of scheduling jobs.
- *Risky mode:* Allocate jobs to any available Grid sites and thus take all possible risks at the resource sites. The risky mode is considered as an aggressive way of scheduling jobs.
- *f-risky mode:* Allocate jobs to available sites to take at most f risk, where f is a probability measure with $f = 0$ for the secure mode and $f = 1$ (i.e., 100%) for the risky mode.

Our preliminary performance study indicated that these two heuristics do perform satisfactorily under various risky conditions to match with user job demands in security assurance. However, their performance is not stable in the perspective of different types of workloads. Furthermore, we developed a new genetic algorithm (GA) to meet two conflicting goals: (1) to have the ability to generate high quality solutions under a security-driven execution model; and (2) to have a low time-complexity so that it can be used in a practical environment. GAs are effective in generating good scheduling solutions [2, 11, 18, 19]. However, GAs are too slow to be used as online scheduling tools. Thus, we propose a novel GA, called the *Space-Time Genetic Algorithm* (STGA). Specifically, in each round of online scheduling, our STGA does *not* start from scratch in that it uses historical scheduling results to generate its chromosome populations. The distinctive feature of this new approach is that the STGA then needs to execute only very few generations to come up with good solutions, making it suitable to be used in an online environment.

Essentially, our new GA performs evolutionary computation not just over "space" (i.e., scheduling solution space) but also over "time" (i.e., its previous scheduling results). We find that this approach makes much sense in a real-life environment because a Grid computing system is expected to receive similar job demands (instead of totally random each time) from the users. Our extensive simulation results indicated that the proposed STGA is highly effective in scheduling two practical workloads: NAS (*Numerical Aerodynamic Simulation*) [13] and PSA (*parameter-sweep applications*) [8], and outperforms the Min-Min and Sufferage heuristics under the three risk modes.

The rest of the paper is organized as follows: In Section 2, we describe in detail our security and risk models and the modified Min-Min and Sufferage heuristics. Section 3 contains the discussions on our proposed new STGA. We provide our extensive simulation results and their interpretations in Section 4. Finally in Section 5, we conclude with some final remarks and suggest future research directions.

2. System Model and Security-Driven Heuristics

We consider a periodic online scheduling system as modeled in Figure 1. Modeling a real-life situation, jobs are accumulated and then scheduled in batches [17]. Thus, the scheduler has to be very efficient such that it will not delay the execution of jobs. We cannot afford to use an offline

algorithm such as simulated annealing [20] or a traditional GA [19]. Jobs are independent in that there is no communication among them. The scheduling objective is to minimize the overall execution time, called the *makespan* which is defined as $\max_i\{\text{FT}(J_i)\}$, where $\text{FT}(J_i)$ is the finish time of job J_i.

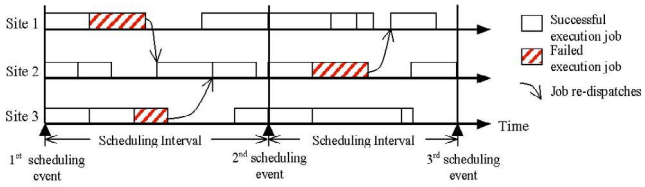


Figure 1. On-line job scheduling system model.

However, a job may fail if it takes the risk of being executed on a site with a security level (SL) lower than its security demand (SD). Specifically, we use the following failure model: The *failure probability* of executing a job, with a job security demand SD on a site with security level SL, is modeled by an exponential distributed failure function as follows:

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

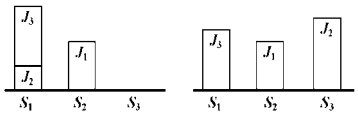
It should be noted that using such a failure model is just for illustration only. We can substitute the above model by any reasonable failure scheme. We also assume fail-stop execution. That is, if a job fails on a site, then it will be rescheduled to restart from the beginning at another site that has an SL greater than the SD, i.e., it is absolutely safe. That is, the scheduler will not allow a failed job to take any risk again.

We consider two well-known heuristics, namely Min-Min and Sufferage [22]. We briefly introduce them below. In the following, ETC (*Expected Time to Complete*) is the expected finish time of a job as determined by the job submission system.

1. **Min-Min heuristic:** For each job, the Grid site that gives the earliest ETC is identified first. Then the job that has the minimum earliest ETC is selected and then assigned to the identified Grid site. For example, as can be seen in Figure 2(a), job J_2 has the smallest value of earliest ETC and thus it gets scheduled first to site S_1. The remaining two jobs J_1 and J_3 are similarly scheduled.

	S_1	S_2	S_3
J_1	3	4	7
J_2	2	4	6
J_3	5	8	13

(a) ETC matrix of execution times in seconds



(b) schedules generated by Min-Min (left: makespan = 7s) and Sufferage (right: makespan = 6s)

Figure 2. A simple scheduling example using the Min-Min and Sufferage heuristics.

2. **Sufferage heuristic:** The Sufferage heuristic is based on the idea that better scheduling results could be generated by assigning a site to a job that would "suffer" most in terms of ETC if that particular site is not assigned to it. Specifically, the sufferage value of a job is the difference between its second earliest completion time (on some site S_y) and its earliest completion time (on some site S_x). That is, using S_x will result in the best completion time, while assigning to S_y will result in the second best. For example, as we can see from Figure 2(a), job J_3 will suffer the most if it does not get scheduled to site S_1, and thus, it is selected first. The other two jobs are similarly scheduled.

We consider three *risk modes* for each of the above two heuristics, according to different risk level experienced. As illustrated in Figure 3, a scheduler is considered as *secure*, if it always schedules a task to a completely safe site (i.e., with SD $<$ SL). That is, it does not allow a task to take any risk. On the other hand, a scheduler is considered as *risky* if it ignores the risk factor completely (i.e., it sets its tolerance of probability of failure to be 1 (100%)). Effectively, the original versions of Min-Min and Sufferage heuristics are considered risky. We also consider an intermediate risk level in that the scheduler will allow a task to take the risk of executing on a site that has SD $<$ SL, provided that the probability of failure is assured to be less than f. We call such a scheduler f-risky.

3. A Fast Space-Time Genetic Algorithm

Genetic algorithms (GAs) have been widely used to tackle many difficult optimization problems. A GA, being a search technique, has a very unique feature–it maintains a pool of potential solutions, called chromosomes, and then tries to "reproduce" new solutions through randomly combining the "good features" of existing solutions. This exploratory searching step is achieved by using the crossover operator, which works by randomly exchanging portions

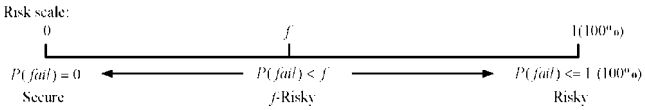


Figure 3. The concept of three risk levels in defining operational modes for security control.

of two chromosomes [19]. Crossover operation searches through the possible solution space. Another important operator is mutation, which works by randomly changing one of the genes in a chromosome. Mutation operation leads the search out of a local optimum, in the hope of getting an even better solution. Finally, there is a selection process–remove the poor solutions (as indicated by their low fitness values) and replace them by the duplicates of the best solutions. In our implementation, 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 [19].

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, in the job scheduling problem, is the completion time of the schedule represented by the solution. In this case, the smallest fitness value represents the best solution. The efficiency of a GA depends largely on the encoding scheme. In the job scheduling problem, the encoding scheme is illustrated in Figure 4.

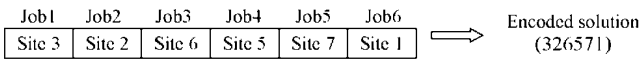


Figure 4. A sample chromosome encoding a possible solution to the job scheduling problem.

Here, each chromosome is an array of numbers with each array index representing a job and the corresponding array element represents the site assignment for that job. The crossover operator is then just a random swapping of two portions of two arbitrarily selected chromosomes. Note that the crossover point is randomly chosen. Mutation is defined as randomly changing the site assignment of a randomly selected job in an arbitrary chromosome to some other site. As in a traditional GA, the crossover and mutation operators are applied as governed by the crossover and mutation probabilities. The whole process is repeated a number of times, which is called the number of generations (iterations).

The crux of our new approach is illustrated in Figure 5. In general, starting from a randomly generated initial pool of solutions (called the initial chromosome population), the quality of the best solution in the pool is usually quite low. Over a number of generations, the solution quality gradually improves. Thus, the question is how we can skip the initial phase of struggling with a pool of not-so-good solutions and start the search closer to the convergence point. Our answer to this question is to make use of *prior scheduling knowledge*–the scheduling results of previous batches.

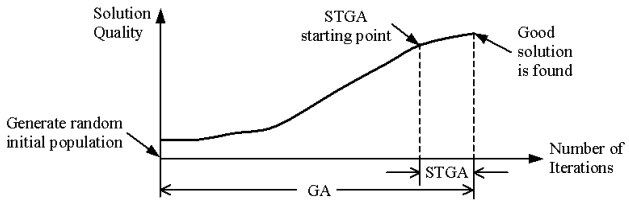


Figure 5. The difference between conventional GA and STGA in terms of solution quality improvement at the starting point on the evolution path.

The justification of this 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 physics researcher trying to generate some simulation data today would very likely try again tomorrow or in the near future. Thus, in our proposed STGA, we keep a history table storing the job specifications and the schedules. Then, such historical data on scheduling could be used to form the initial population when the GA is invoked.

Specifically, in each entry of the lookup table, the input of each entry contains three parameters: (1) the next available times of sites, (2) job execution time matrix, and (3) job security demands. The similarity between the new input jobs and each entry is the average similarity for the three parameters. We map each parameter into a one-dimensional vector, and then we use the following vector comparison method to calculate the similarity for each parameter. Suppose two k-element row vectors $\mathbf{a} = (a_1, a_2, \ldots, a_k)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_k)$, the similarity between vectors $\mathbf{a}$ and

$\mathbf{b}$ is given by:

$$\text{Similarity}(\mathbf{a}, \mathbf{b}) = 1 - \frac{\sum_{i=1}^{k} \|a_i - b_i\|}{\max\{\max\{a_i\}, \max\{b_i\}\}} \quad (2)$$

Because such previous scheduling results were good solutions to similar problems, the initial population formed will then be of a moderately high solution quality. Thus, our proposed STGA does not only perform evolutionary searching over "space" (i.e., the solution space as manifested by the chromosome population) but also over "time" (i.e., the historical scheduling result table). The overall scheduling mechanism of our proposed STGA is illustrated in Figure 6. The initial populations of the STGA also include those randomly generated solutions to guarantee enough diversity. In the simulation experiments, we use the Min-Min and Sufferage heuristics to a fixed number of training jobs to generate the initial lookup table entries. In real-life applications, the initial lookup table could be built from the beginning. The LRU (*Least Recently Used*) algorithm is adopted to update the entries in the lookup table.

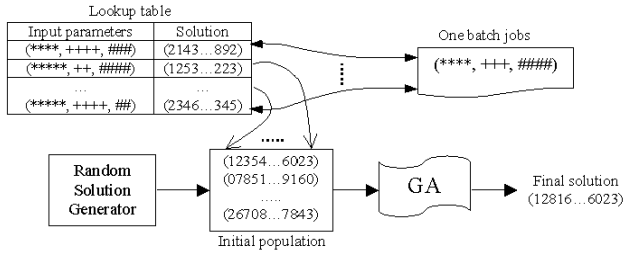


Figure 6. The proposed new space-time genetic algorithm (STGA) for trusted online scheduling.

4. Simulation Results over the NAS and PSA Workloads

In this section, we first describe in detail the performance metrics that we have used in evaluating the effectiveness of the scheduling algorithms considered in our study (i.e., the three versions of the Min-Min and Sufferage heuristics, and our proposed STGA). We then introduce two types of workloads and present the simulation results obtained.

4.1. Performance Metrics

To comprehensively evaluate the scheduling performance, we have used the following metrics:

- *Average response time*: Denote the total number of simulated jobs as N, and denote the completion time for a single job J_i as c_i, the arrival time as a_i, the average response time is defined as $\frac{\sum_{i=1}^{N}(c_i - a_i)}{N}$.
- *Makespan*: Defined as $\max\{c_i\}$.
- *Slowdown ratio*: Denote the start time for a single job J_i as b_i, the average waiting time is defined as: $\frac{\sum_{i=1}^{N}(c_i - b_i)}{N}$. The slowdown ratio is defined as the ratio between the average response time and the average waiting time. This metric indicates the average contention experienced by a job. That is, we have:

$$\text{Slowdown Ratio} = \frac{\sum_{i=1}^{N}(c_i - a_i)/N}{\sum_{i=1}^{N}(c_i - b_i)/N} \quad (3)$$

- *Number of risk-taking jobs* N_{risk}: When sites provided security level cannot satisfy the jobs security demand, N_{risk} counts the number of jobs that are running on such kinds of sites.
- *Number of failed jobs* N_{fail}: Job execution may fail owing to insecure resource sites applied. N_{fail} counts the number of failed and rescheduled jobs. N_{fail} is bounded above by N_{risk}.
- *Site utilization*: Defined by the percentage of processing power allocated to user jobs out of total processing power available at a selected Grid site.

4.2. The NAS and PSA Workloads

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

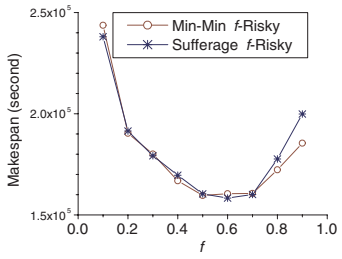
NAS trace workloads: We use three months accounting records for the 128-node iPSC/860 located in the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center. This trace contains 92 days of data, gathered in year 1993. For testing the performance of the heuristics under a high-throughput Grid environment, the 92 days trace data is proportionally squeezed to 46 days. We map the 128 nodes to 12 Grid sites–four of the sites each contain 16 nodes, and the other eight sites each contain 8 nodes. Our simulations are based on the arrival time, job size, and runtime data provided by the trace [21]. This trace was sanitized to remove the user specified information and pre-processed to correct for system downtime. Detailed information about the characteristics of the trace can be found in [13].

Parameter-sweep application (PSA) workloads: The parameter-sweep application is defined as a set of independent sequential jobs (i.e., no job precedence) [8]. The independent jobs operate on different datasets. A range of scenarios and parameters to be explored are applied to the pro-

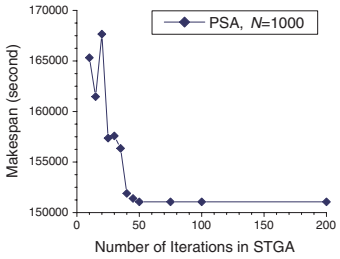
gram input values to generate different data sets. The execution model essentially involves processing N independent jobs (each with the same task specification, but a different dataset) on M distributed sites where N is, typically, much larger than M. Due to space limitations, we can only show the scalability results for the PSA workloads.

4.3. Simulation Parameters and Settings

To address the following two questions: (1) In f-risky scheduling, why we choose $f = 0.5$ (50%)? (2) In the STGA scheduling, why we choose 100 as the total number of iterations? We analyze below in Figure 7 the makespan variation of the f-risky scheduling as f increases from 0 to 1. Similarly, we show the makespan variation of the STGA algorithm, as the total number of iterations increases.



(a) Makespan of f-Risky heuristics



(b) Makespan of STGA

Figure 7. Plots of makespan for two f-risky heuristics against increasing risk level f and of the STGA against increasing iterations under the PSA workloads over 1,000 jobs.

Figure 7(a) shows the makespan results of Min-Min f-risky and Sufferage f-risky algorithms as a function of f. Two concave curves are observed. The f-Min-Min achieves its minimum at $f = 0.5$. The f-Sufferage achieves its minimum at $f = 0.6$. The following assertion is made from this analysis: The optimal f value to achieve the minimum makespan is in the range of 0.5–0.6, or very close to this range. Thus, we choose $f = 0.5$ in all of our f-risky heuristic simulation experiments. Figure 7(b) shows the makespan of the STGA scheduling as a function of the number of evolution iterations. The makespan fluctuates in the first 25 iterations, starts to converge at 40 iterations, and converges to a flat constant value after 50 iterations. This demonstrates the fastness of our STGA. In the rest of our experiments, we choose a larger number of iterations, say 100 iterations.

Table 1 lists important simulator parameters from the two benchmark workloads, most of the configuration and workload data supplied by the provider agencies.

Parameter	Value
Number of jobs	NAS: 16000; PSA: 5000
Number of sites	NAS: 12; PSA: 20
Job arrival rate	NAS: Given by trace; PSA: 0.008
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)
Site security level	0.4–1.0 (uniform dist.)
Job security demand	0.6–0.9 (uniform dist.)
Number of generations	100
Initial population size	200
Crossover probability	0.8
Mutation probability	0.01
Lookup table size	150
Number of training jobs	500
Similarity threshold	0.8

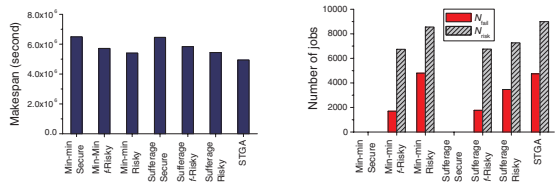
Table 1. Simulation parameters and their values

4.4. Results over NAS Trace Workloads

In this section, we evaluate the performance of the seven heuristic algorithms on the NAS trace workloads. The evaluation is based on the performance metrics listed in Section 4.1. The simulation results are given for each metric, followed by a statistical analysis over all metrics.

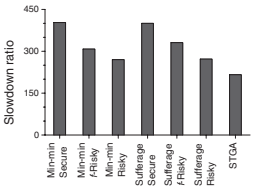
Figure 8(a) shows the makespan results of the seven heuristic algorithms. Overall, the STGA has the best performance. The two secure mode algorithms perform the worst.

Risky algorithms perform slightly better than the f-risky ones. Roughly 10% improvement of STGA compared with risky algorithms is observed. Roughly 15% improvement of STGA is observed, compared with the f-risky algorithms. A much higher 30% improvement is observed in using the STGA, compared with using the secure algorithms. Figure 8(b) shows the number of failed jobs and number of risk-taking jobs. Under a secure mode, inherently, there is no failure. For the other five algorithms, the STGA and Min-Min risky algorithms have the largest number of risk-taking jobs and the largest number of failed jobs. The number of failed jobs for f-risky algorithms is half of that of the risky heuristics. Overall, the numbers of risk-taking jobs are comparable for all algorithms.

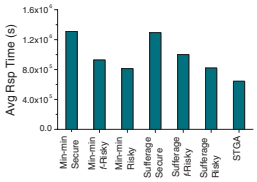


(a) Makespan in seconds

(b) Numbers of failed and risk-taking jobs



(c) The slowdown ratio



(d) Average response time in seconds

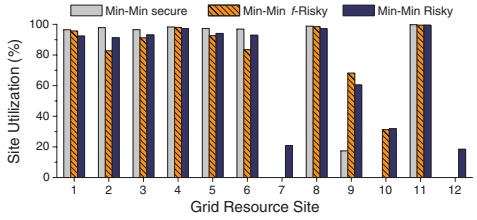
Figure 8. Performance results of 6 heuristic algorithms and of the STGA for the NAS trace workload.

Figure 8(c) shows the slowdown ratio results. STGA has the minimum slowdown ratio. We calculate the performance improvement factors of the STGA, compared with other algorithms. The STGA has more than 46% improvement compared with two secure heuristics. STGA has more than 35% improvement compared with the f-risky heuristics. The STGA has more than 20% improvement over the risky heuristics. Figure 8(d) shows the average response times. The STGA has the shortest response time. The two secure heuristics have the longest response times. The response times of two risky heuristics are shorter than those of the two f-risky heuristics. Overall, the STGA has roughly improved 50% over the two secure heuristics. The STGA has roughly 30% improvement over the two f-risky heuristics. The STGA improves by 20%, compared with the risky heuristics.

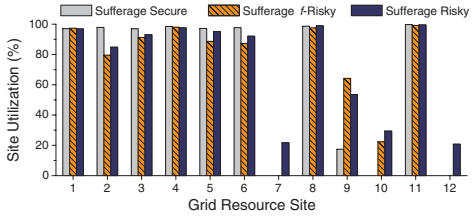
Figure 9 shows the site utilization rate. For all secure, f-risky and risky heuristics, the utilization rate has almost no difference between Min-Min and Sufferage heuristics. As indicated by the results of the secure mode algorithms, the utilization is not balanced among various Grid sites. Some sites are simply not used. Eight sites among the 12 sites achieved very high (more than 95%) utilization rates, while one site has a much lower utilization rate, roughly 20%. The other 3 sites have not been used at all.

On the other hand, we can see from the results of the two f-risky heuristics that their utilization rates are much balanced among the sites. Fewer sites are not used compared with the secure heuristics. Among the 12 sites, 8 sites achieve high utilization rates, 2 sites achieve relatively lower utilization rates, and the other 2 sites have not been used. For the results of the two risky heuristics, owing to the aggressive nature of this operational mode, no idle site is observed. Thus, the site usages are much better balanced. Compared with the f-risky heuristics, more sites achieved higher utilization rates. The utilization rate of the STGA is shown in Figure 9(c). The STGA has the most balanced utilization. No idle site is observed, and more sites achieve high utilization rates. With reference to the utilization rates of three best-performed algorithms in Figure 9(c), STGA still achieves slightly higher utilizations compared with the risky heuristics.

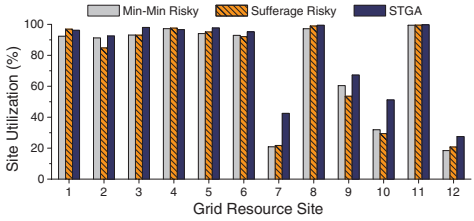
A global performance comparison of the 6 security-driven heuristics and the STGA algorithm is shown in Table 2 under the NAS trace workload. The relative ranking is determined by comparing the makespan and average response time of each algorithm with respect to those of the STGA. Specifically, we define α to be the *makespan ratio* of a heuristic with respect to the STGA. Similarly, we define β to be the *response time ratio* of a heuristic with respect to the STGA. Considering all the performance metrics in a holistic manner, the STGA is the best. The two risky algorithms have the second place. The two f-risky heuristics rank the third. The two secure heuristics have the lowest among the seven. We conclude that our STGA is the most effective scheduling algorithm when real-life workloads like NAS and PSA are applied.



(a) Min-Min



(b) Sufferage



(c) An overall comparison of the three algorithms

Figure 9. Site utilization (%) for the NAS trace workloads.

4.5. Scaling Effects of Workload Size (Number of Jobs, N)

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 number of simulated jobs are reported in Figure 10. Because the number of jobs in the NAS trace workload is fixed, we run experiments only on the PSA workloads. The number of simulated jobs varies as N = 1000, 2000, 5000, and 10000.

Heuristics		α	β	Ranking
Min-Min	Secure	1.314	2.035	4th
	0.5-Risky	1.157	1.441	3rd
	Risky	1.094	1.262	2nd
Sufferage	Secure	1.307	2.011	4th
	0.5-Risky	1.181	1.555	3rd
	Risky	1.102	1.275	2nd
STGA		1.000	1.000	1st

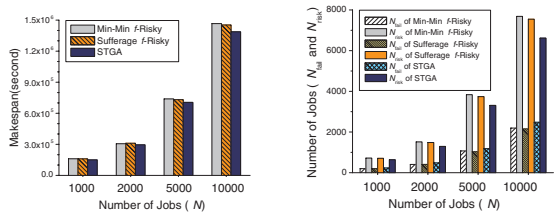
Table 2. Performance comparison of the 6 Heuristics and the STGA on NAS trace workloads

Min-Min f-risky, Sufferage f-risky and STGA are the three best-performed scheduling algorithms for the PSA workloads. Thus, the scalability analysis is based on these three algorithms. Overall, all measured performance metrics show a monotonic increasing trend. STGA has the best performance. The results of Min-Min f-risky and Sufferage f-risky are comparable for all N. The Sufferage f-risky performs slightly better than the Min-Min f-risky. The performance improvement is within 1%.

Figure 10(a) shows the makespan of these three algorithms. There is 6% improvement of STGA compared with Min-Min f-risky and Sufferage f-risky. Figure 10(b) shows both the number of failed jobs and risk-taking jobs. Again, STGA has fewer jobs taking risks but has more jobs failed compared with Min-Min f-risky and Sufferage f-risky. One explanation for this phenomenon is that the f-risky algorithms set the risk threshold f = 0.5. Thus, even more jobs are taking risks, fewer jobs fail during execution. Figure 10(c) shows the slowdown ratios of these three algorithms. Figure 10(d) shows the average response times. There is roughly 40% improvement of STGA compared with the other two algorithms for both slowdown ratio and average response time.

5. Conclusions and Future Research

Security-driven job scheduling is crucial to achieving high performance in an open Grid computing environment. However, existing scheduling algorithms largely ignore the security induced risks involved in dispatching jobs to remote sites, which are owned by other organizations. In our study, we first devise a security- driven scheduling model and then we modify two well-known heuristics to work under our new model, which captures various risk levels in real-life applications. Under the risky conditions, our security-driven Min-Min and Sufferage heuristics do perform well to match the user security demand with the site



(a) Makespan in seconds

(b) Numbers of failed and risk-taking jobs

(c) The slowdown ratio

(d) Average response time in seconds

Figure 10. Performance results of 6 heuristic algorithms and the STGA for the PSA workloads with number of jobs N = 1000, 2000, 5000, and 10000.

security assurance provided. However, their performance is not stable when applying to different types of workloads.

Genetic algorithms were not applied to Grid scheduling in the past for their slowness in generating good solutions. We propose a new Space-Time Genetic Algorithm (STGA), which works by swiftly generating good solutions based on a pool of previously found solutions. This new STGA algorithm is designed to meet the two conflicting goals: (1) capable of generating high quality solutions, and (2) capable of quickly generating solutions.

Based on a statistical analysis, we find that the makespan of the STGA algorithm is lower bounded by a minimum time needed to evolve from sufficient crossover and mutation operations. The minimum time can be reached after 50 evolution iterations in testing on the PSA workloads. Thus, we suggest training the STGA process for 100 iterations before using the algorithm for scheduling purpose. By doing so, we have accelerated the STGA solution time significantly. This translates to the low overhead experiences in on-line scheduling of large number of jobs.

Our extensive simulation results indicate that all the three algorithms scale well with the increase of the total job number. For example, to schedule 10,000 jobs with the NAS trace workload, the makespan of all 7 algorithms are approximately equal in magnitude. The slowdown ratio and the average job response time of the STGA algorithm is only 60% of those required in using the Min-Min and Sufferage heuristics. The only drawback in using the STGA is its high number of risk-taking jobs experienced. Given the promising performance of the STGA observed by our Grid scheduling simulation experiments, we are now working on the production prototype scheduler based on the security-driven heuristics and STGA proposed. Apart from this major future research avenue, we believe that investigating the performance of the STGA, when the job execution durations are unknown a priori is also an important problem [15].

Acknowledgments

The authors gratefully acknowledge the funding support of this work by an NSF ITR Grant under contract number ACI-0325409. This work was conducted at the USC Internet and Grid Computing Laboratory. The authors very much appreciate the critical comments from the technical members of the USC GridSec research group and the anonymous IPDPS 2005 reviewers.

References

[1] J. H. Abawajy, "Fault-Tolerant Scheduling Policy for Grid Computing Systems," *Proc. IPDPS 2004*.

[2] A. Auyeung, I. Gondra, and H. K. Dai, "Multi-Heuristic List Scheduling Genetic Algorithm for Task Scheduling," *Proc. ACM Sym. Applied Computing 2003*, pp. 721–724.

[3] F. Azzedin and M. Maheswaran, "Integrating Trust into Grid Resource Management Systems," *Proc. ICPP 2002*.

[4] M. A. Baker, R. Buyya, and D. Laforenza, "The Grid: International Efforts in Global Computing," *International Journal of Software Practice and Experience*, vol. 32, no. 15, Nov. 2002.

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

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

[7] T. D. Braun, H. J. Siegel *et al.*, "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, 2001, pp. 810–837.

[8] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," *Proc. HCW 2000.*

[9] K. Cooper *et al.*, "New Grid Scheduling and Rescheduling Methods in the GrADS Project," *Proc. IPDPS 2004*.

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

[11] V. Di Martino, "Sub-Optimal Scheduling in a Grid Using Genetic Algorithms," *Proc. IPDPS 2003*.

[12] 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, Mar. 2002, pp. 308–323.

[13] 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.

[14] N. Fujimoto and K. Hagihara, "Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid," *Proc. ICPP 2003*.

[15] M. Harchol-Balter, "Task Assignment with Unknown Duration," *J. of ACM*, vol. 49, no. 2, Mar. 2002, pp. 260–288.

[16] M. Humphrey and M. R. Thompson, "Security Implications of Typical Grid Computing Usage Scenarios," *IEEE Proc. HPDC 2001*, pp. 95–103.

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

[18] S. Kim and J. B. Weissman, "A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications," *Proc. of ICPP 2004*.

[19] Y.-K. Kwok and I. Ahmad, "Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using A Parallel Genetic Algorithm," *J. Parallel and Distributed Computing*, vol. 47, no. 1, pp. 58–77, Nov. 1997.

[20] Y.-K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

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

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

[23] S. Song, K. Hwang, and M. Macwan, "Fuzzy Trust Integration for Security Enforcement in Grid Computing," *Proc. of IFIP Int'l Conf. on Network and Parallel Computing*, Oct. 2004.

[24] S. Song, K. Hwang, and Y.-K. Kwok, "Security Binding for Trusted Job Outsourcing in Open Computational Grids," submitted to *IEEE Trans. Parallel and Distributed Systems*.

[25] X.-H. Sun and M. Wu, "Grid Harvest Service: A System for Long-Term, Application-Level Task Scheduling," *Proc. of IPDPS 2003*.

[26] M. Wu and X.-H. Sun, "A General Self-Adaptive Task Scheduling System for Non-Dedicated Heterogeneous Computing," *Proc. of Cluster Computing (Cluster 2003)*.

Authors' Biographies

Shanshan Song received her B.S. degree in Computer Science from Special Class for Gifted Young in University of Science and Technology of China in 2001. She is currently pursuing the Ph.D. degree in the Department of Computer Science at University of Southern California. 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 can be reached at *shanshan.song@usc.edu* or via the URL: http://www-scf.usc.edu/ shanshas/.

Yu-Kwong Kwok received the B.S. degree in Computer Engineering from the University of Hong Kong (HKU), in 1991, and the M.Phil. and Ph.D. degrees in Computer Science from the Hong Kong University of Science and Technology (HKUST), in 1994 and 1997, respectively. He is an Associate Professor in the Department of Electrical and Electronic Engineering at HKU. Dr. Kwok is currently on leave from HKU and is a Visiting Associate Professor at the University of Southern California. His research interests include Grid computing, mobile computing, wireless communications, network protocols, and distributed computing algorithms. Dr. Kwok is a member of the Association for Computing Machinery (ACM), the IEEE Computer Society, and the IEEE Communications Society. He is a Senior Member of the IEEE. He can be reached at *ykwok@hku.hk*. He is a recipient of the 2003-2004 Outstanding Young Researcher Award at HKU.

Kai Hwang is a Professor and Director of Internet and Grid Computing Laboratory at the University of Southern California. He received the Ph.D. from the University of California, Berkeley. An IEEE Fellow, he specializes in computer architecture, parallel processing, Internet and wireless security, Grid and cluster computing, and distributed computing systems. He has authored or coauthored 7 scientific books and over 180 Journal and Conference papers in these areas. Dr. Hwang is the founding Editor-in-Chief of the Journal of Parallel and Distributed Computing. He is also on the editorial board of IEEE Transactions on Parallel and Distributed Systems. He has performed advisory and consulting work for IBM Fishkill, Intel SSD, MIT Lincoln Lab., ETL in Japan, and GMD in Germany. Presently, he leads the NSF-supported ITR GridSec project at USC. The GridSec group develops security-binding techniques for trusted Grid computing. The group builds self-defense software systems for protecting Grid and distributed computing resources. Dr. Hwang can be reached at USC via *kaihwang@usc.edu* or through the URL: http://GridSec.usc.edu/Hwang.html.