Enhancing Security of Real-Time Applications on Grids through Dynamic Scheduling

Tao Xie Xiao Qin*
Department of Computer Science
New Mexico Institute of Mining and Technology
801 Leroy Place, Socorro, New Mexico 87801-4796
{xietao, xqin}@cs.nmt.edu

Abstract

Real-time applications with security requirements are emerging in various areas including government, education, and business. The security sensitive real-time applications can take full advantage of a grid environment that allows grid participants to exercise a fine-grained control and allocation of computational resources. However, conventional real-time scheduling algorithms failed to fulfil the security requirements of real-time applications. In this paper we propose a dynamic real-time scheduling algorithm, or SAREG, which is capable of enhancing quality of security for real-time applications running on Grids. To make SAREG practical, we present a mathematical model to formally describe a scheduling framework, security-sensitive real-time applications, and security overheads. We leverage the model to measure security overheads incurred by an array of security services, including encryption, authentication, integrity check, etc. The SAREG algorithm seamlessly integrates security requirements into real-time scheduling by employing the security overhead model. To evaluate the effectiveness of SAREG, we conducted extensive simulations using a real world trace from a supercomputing centre. Our experimental results show that SAREG significantly improves system performance in terms of quality of security and schedulability over three existing scheduling algorithms.

1. Introduction

A computational grid is a collection of geographically dispersed computing resources, providing a large virtual computing system to users. With rapid advances in processing power, network bandwidth, and storage capacity, Grids are emerging as next generation computing platforms for large-scale computation and data intensive problems in industry, academic, and government organizations. As typical scientific simulation and computation require a large amount of compute power, it becomes crucial to take advantage of application scheduling to enable the sharing and aggregation of geographically distributed resources to meet the needs of highly complex scientific problems [32].

Recently there have been some efforts devoted to the development of real-time applications on Grids [9]. Real-time applications depend not only on results of computation, but also on time instants at which these results become available [13]. The consequences of missing deadlines of hard real-time systems may be catastrophic, whereas such consequences for soft real-time systems are relatively less damaging. Examples of hard real-time applications include distributed defense and surveillance applications [38], and distributed medical data systems [20]. On-line transaction processing systems are examples of soft real-time applications [26].

There exist a growing number of systems that have real time and security considerations [36], because sensitive data and processing require special safeguard and protection against unauthorized access. In particular, a variety of motivating real-time applications running on Grids require security protections to completely fulfill their security-critical needs. Unfortunately, conventional wisdom on real time systems is inadequate for applications with real-time and security requirements. This is mainly because traditional real-time systems are developed to guarantee timing constraints while possibly posing unacceptable security risks.

To tackle the aforementioned problem, we proposed a real-time scheduling algorithm (referred to as SAREG) with security awareness, which is intended to seamlessly integrate security into real-time scheduling for applications running on Grids. In this paper we use trace-driven simulation to compare the performance of the SAREG algorithm against three baseline scheduling algorithms for Grids. Our simulator combines performance and security overhead estimates using a security overhead model based on three most commonly used security services. We have used a real world trace from a supercomputing centre to drive our simulations. Also, we concentrate on three security services, namely, authentication, integrity, and encryption. Our empirical results demonstrate that the SAREG scheduling algorithm

* Contact author. *http://www.cs.nmt.edu/~xqin*

is able to achieve high quality of security while guaranteeing timing constraints posed by real-time applications.

To put our work in a large perspective, in the next section we summarize related work in the areas of computer security and real-time systems. Section 3 describes the preliminary system architecture and task model. In Section 4 we propose a real-time scheduling algorithm for parallel applications on Grids. We present in Section 5 the experimental results based on real world traces from a supercomputing centre. We also provide insight into system parameters that ultimately affect performance potential of SAREG. Finally, Section 6 concludes the paper with summary and future directions.

The rest of the paper is organized in the following way. Section 2 includes a summary of related work in this area. Section 3 discusses the system architecture and task model with security requirements. Section 4 proposes a security overhead model. Section 5 presents the security-aware real-time scheduling strategy. Performance analysis of the SAREC-EDF algorithm is explained in Section 6. Section7 concludes the paper with summary and future research directions.

2. Related work

Scheduling algorithms for Grids have been extensively studied in the past both experimentally and theoretically. Wu and Sun considered memory availability as a performance factor and introduced memory-aware scheduling in Grid environments [42]. Li compared the performance of a variety of scheduling and processor allocation algorithms for grid computing [21]. In et al. proposed a framework for policy based scheduling in resource allocation of grid computing [18]. However, these scheduling algorithms are not suitable for real-time applications, because there is no guarantee to finish real-time tasks in specified time intervals.

The issue of scheduling for real-time applications was previously reported in the literature. Conventional real-time scheduling algorithms such as Rate Monotonic (RM) algorithm [23], Earliest Deadline First (EDF) [37], and Spring scheduling algorithm [33] were successfully applied in real-time systems. We proposed both static [30] and dynamic [31] scheduling schemes for real-time systems. Unfortunately, none of the above real-time scheduling algorithms can be directly applied to the Grid environments.

Real-time scheduling is a key factor in obtaining high performance and predictability for Grids. Various aspects of complicated real-time scheduling problems in the context of Grids were addressed in the literature. He et al. proposed a dynamic scheduling for parallel jobs with soft-deadlines in Grids [16]. Caron et al. developed an algorithm that considers both priority and deadlines of tasks on Grids [6]. However, most of existing real-time scheduling algorithms perform poorly for security-sensitive real-time applications on Grids due to the ignorance of security requirements imposed by the applications.

Recently increasing attention has been drawn toward security-awareness in Grids, because efficient and flexible security has become a baseline requirement. Humphrey et al. examined the current state of the art in securing a group of activities and introduced new technologies that promise to meet the security requirements of Grids [17]. Azzedin and Maheswaran integrated the notion of "trust" into resource management of a grid computing systems [3]. Wright et al. proposed a security architecture for a network of computers bound together by an overlying framework used to provide users a powerful virtual heterogeneous machine [41]. Connelly and Chien proposed an approach to protecting tightly coupled, high-performance component communication [7]. However, the above security techniques are not appropriate for real-time applications due to the lack of ability to express and handle timing constraints.

Some work has been done to incorporate security into a variety of real-time applications. George and Haritsa proposed concurrency control protocols to support applications with real-time and security requirements [12]. Ahmed and Vrbsky developed a secure optimistic concurrency control protocol that can make trade-offs between security and real-time requirements [2]. Son et al. proposed an approach to trading off quality of security to achieve required real-time performance [35]. In [36], a new scheme was developed to improve timeliness by allowing partial violations of security. Our work is fundamentally different from the above approaches because they are focused on concurrency control protocols whereas ours is intended to develop a security-aware real-time scheduling algorithm, which can meet security constraints in addition to real-time requirements of tasks running on Grids.

Most recently, we proposed a dynamic security-aware scheduling algorithm for a single machine [43] and clusters [44]. We conducted simulations to show that compared with three heuristic algorithms, the proposed algorithm can consistently improve overall system performance in terms of quality of security and system schedulability under a wide range of workload conditions.

3. Mathematical Model

A mathematical model of security-aware real-time scheduling for Grids is presented in this section. This model, which describes a scheduling framework, security-sensitive real-time jobs, and security overheads, allows the SAREG algorithm to be formally presented in Section 4.

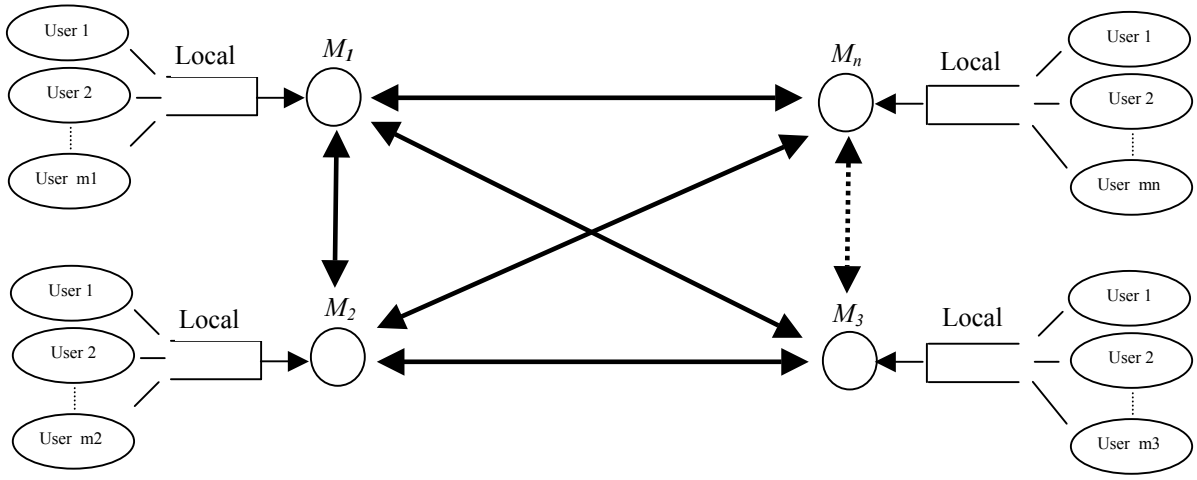


Figure 1. The Scheduling Framework for SAREG in a computational Grid.

3.1. Scheduling Framework

A Grid can be specified as $G = \{M_1, M_2, \ldots, M_n\}$, where M_i, $1 \le i \le n$, is a site or cluster [27]. The n sites are connected by wide-area networks (See Figure 1). The scheduling framework is general in the sense that it can be applied to small-scale grids where computational sites are connected by LAN or MAN. Each site M_i is represented as a vector, e.g., $M_i = (P_i, N_i, T_i, Q_i)$, where P_i is the peek computational power measured by an overall CPU capacity (e.g., BIPS), N_i is the number of machines in the site, T_i is a set of accepted jobs running on M_i, and Q_i is a scheduler. Note that there exists a scheduler in each site, and we advocate the use of a distributed scheduling framework rather than a centralized one. This is mainly because a centralized scheduler in a large-scale grid inevitably becomes a severe performance bottleneck, resulting to a significant performance drop when workload is high. The distributed scheduling infrastructure makes a system portable, secure, and capable of distributing scheduling workload among an array of computational sites in the system [28][29].

Each site continues to receive reasonably up-to-date global load information by monitoring resource utilization of the Grid, and periodically broadcasts it local load information to other sites of the Grid. When a real-time job is submitted by a user to a local site, the corresponding scheduler assigns the job to a group of local machines or migrate the job to a remote site within in the Grid. The scheduler consists of a s*chedule queue* used to accommodate incoming real-time jobs. The scheduler queue is maintained by an admission controller. If the incoming real-time jobs can be scheduled, the admission controller will place the tasks in the *accepted queue* for further processing. In case no site can guarantee the deadline of the submitted real-time job, it will be dropped into a local *rejected list*. The scheduler processes all the accepted tasks by its scheduling policy before transmits them into the *dispatch queue*, where the quality of security of accepted jobs are maximized. After the quality of security is enhanced, the real-time job is dispatched to one of the designated site $M_i \in G$. The machines in site M_i can execute a number of real-time tasks in parallel.

3.2. Security-Sensitive Real-time Jobs

A real-time job is specified as a set of rational parameters, e.g., $J_i = (e_i, p_i, d_i, l_i, S_i)$, where e_i is the execute time, p_i is the number of machines required to execute J_i, d_i is the deadline, and l_i denotes the amount of data (measured in KB) to be protected. e_i can be estimated by code profiling and statistical prediction [5]. A collection of security services required by J_i is specified as $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, where S_i^j denotes the security level range of the jth security service. Our security-aware scheduler is intended to determine the most appropriate point s_i in space S_i, e.g., $s_i = (s_i^1, s_i^2, \ldots, s_i^q)$, where $s_i^j \in S_i^j$, $1 \le j \le q$.

A schedule of a job J_i is formally denoted as the following expression:

$$x_i = \left((\sigma_{i,1}, s_{i,1}), (\sigma_{i,2}, s_{i,2}), \ldots (\sigma_{i,pi}, s_{i,pi})\right), \qquad (1)$$

where J_i is divided into p_i tasks, $\sigma_{i,j}$ and $s_{i,j}$ are the start time and the security level of the jth task.

The SAREG algorithm is able to measure the security benefits gained by each admitted job. To implement this basic and valuable functionality, we quantitatively model the security benefit of the jth task of job J_i as a security

level function denoted by SL: $S_i \rightarrow \Re$, where $\Re$ is the set of positive real numbers:

$$SL(s_{i,j}) = \sum_{k=1}^{q} w_i^k s_{i,j}^k , \quad (2)$$

where $s_{i,j} = \left(s_{i,j}^1, s_{i,j}^2, ..., s_{i,j}^q\right), 0 \le w_i^j \le 1, \sum_{j=1}^{q} w_i^j = 1$

Note that w_i^j is the weight of the jth security service for the task. Users specify in their requests the weights to reflect relative priorities given to the required security services. The security benefit of job J_i is computed as the summation of the security levels of all its tasks. Thus, we have the following equation:

$$SL(s_i) = \sum_{j=1}^{p_i} SL(s_{i,j}), \quad (3)$$

where $s_i = (s_{i,1}, s_{i,2}, ..., s_{i,pi})$.

The scheduling decision of the job J_i is feasible if (1) all its tasks can be completed before the deadline d_i, and (2) the corresponding security requirements are satisfied. Specifically, given a real-time job J_i that consists of p_i tasks, we can obtain the following non-linear optimization problem formulation to compute the optimal security benefit of J_i:

$$\text{maximize } SL(s_i) = \sum_{j=1}^{p_i} \sum_{k=1}^{q} w_i^k s_{i,j}^k , \quad (4)$$

subject to $\min(S_i^k) \le s_{i,j}^k \le \max(S_i^k), \quad f_{i,j} \le d_i$

where $f_{i,j}$ is the finish time of the jth task of J_i, and $\min(S_i^j)$ and $\max(S_i^j)$ are the minimum and maximum security requirements of J_i.

The SAREG scheduling algorithm to be presented in the next section strives to enhance quality of security, which is defined by the sum of the security levels of all the admitted jobs. Thus, the following security value function needs to be optimized, subjecting to certain timing and security constraints:

$$\text{maximize } SV = \sum_{j=1}^{n} \sum_{J_i \in T_j} y_{i,j} SL(s_i), \quad (5)$$

A where T_j is a set of accepted jobs running on site M_j, $y_{i,j}$ is set to 1 if job J_i is accepted by the jth site, and is set to 0 otherwise. Substituting Equation (4) into (5) yields the following security value objective function. Thus, our job scheduling problem for Grid environments can be formally defined as follows: given a Grid $G = \{M_1, M_2, ..., M_n\}$ and a list of jobs submitted to the Grid, find a schedule $x_i = \left((\sigma_{i,1}, s_{i,1}), (\sigma_{i,2}, s_{i,2}), ...(\sigma_{i,pi}, s_{i,pi})\right)$ for each job J_i, such that the total security level of jobs on G,

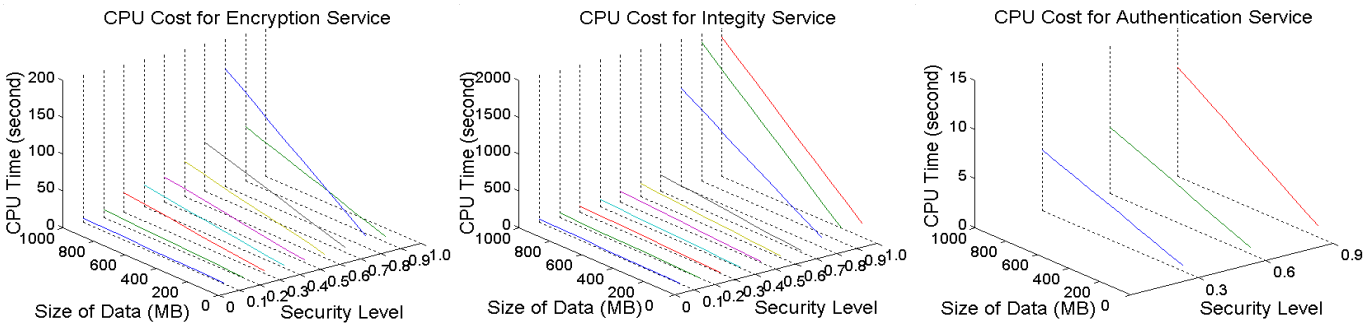
$$SV = \sum_{i=1}^{n} \sum_{J_j \in T_i} y_{i,j} \sum_{k=1}^{p_j} \sum_{l=1}^{q} w_j^k s_{j,k}^l , \quad (6)$$

is maximized.

3.3. Security Overhead

Since security is achieved at the cost of performance degradation, it is critical and fundamental to quantitatively measure overhead caused by various security services. Unfortunately, less attention has been paid to models used to measure security overheads. Recently Irvine and Levin proposed a security overhead framework, which can be used for a variety of purposes [19]. However, security overhead model for each security services in the context of real-time computing remains an open issue. To enforce security in real-time applications while making security-aware scheduling algorithms predictable and practical, in this section we proposed an effective model that is capable of approximately, yet reasonably, measuring security overheads experienced by tasks with an array of security requirements. With the security overhead model in place, schedulers are enabled to be aware of security overheads, thereby incorporating the overheads into the process of scheduling tasks. Particularly, the model can be employed to compute the earliest start times and the minimal security overhead (see Equation 12 and Equation 13).

Without loss of generality, we consider three security services widely deployed in real-time systems, namely, encryption, integrity, and authentication.



(a) Encryption **(b) Integrity** **(c) Authentication**

Figure 2. CPU overhead of security services.

- **Encryption Overhead**

Encryption is used to encrypt real-time applications (executable file) and the data they produced such that a third party is unable to discover users' private algorithms embedded in the executable applications or understand the data created by the applications. Suppose each site has ten optional encryption algorithms, which are listed in Table 1. Based on their performance, each cryptographic algorithm is assigned a corresponding security level in the range from 0.1 to 0.9. For example, level 0.9 implies that we use 3DES, which is the strongest yet slowest encryption function among the alternatives. Since computation overhead caused by encryption mainly depends on the cryptographic algorithms used and the size of the data to be protected, Figure 2 (a) shows encryption time in seconds as a function of encryption algorithms and size of data to be secured measured on a 175 MHz Dec Alpha600 machine [25].

Table 1. Cryptographic Algorithms Used for Encryption Service

Cryptographic	s_i^e : SL	$\mu^e(s_i^e)$:MB/s
SEAL	0.1	168.75
RC4	0.2	96.43
Blowfish	0.3	37.5
Knufu/Khafre	0.4	33.75
RC5	0.5	29.35
Rijndael	0.6	21.09
DES	0.7	15
IDEA	0.8	13.5
3DES	0.9	6.25

Let s_i^e be the encryption security level of t_i, and the computation overhead of the encryption service can be calculated using Equation (7), where l_i is the amount of data whose confidentiality must be guaranteed, and $\mu^e(s_i^e)$ is a function used to map a security level to its corresponding encryption method's performance.

$$c_i^e(s_i^e) = l_i / \mu^e(s_i^e). \quad (7)$$

- **Integrity Overhead**

Integrity services make it possible to ensure that no one can modify or tamper applications while they are executing on clusters. This can be accomplished by using a variety of hash functions [4]. Ten commonly used hash functions and their performance (evaluated on a 90 MHz Pentium machine) are shown in Table 2. Based on their performance, each hash function is assigned a corresponding security level in the range from 0.1 to 1.0. For example, level 0.1 implies that we use MD4, which is the weakest yet fastest hash function among the alternatives. Level 1.0 means that Snefru-256 is employed for integrity, and Snefru-256 the slowest yet strongest function among the competitors.

Table 2. Ten Hash Functions Used for Integrity Service

Hash	s_i^g : SL	$\mu^g(s_i^g)$:KB/ms
MD4	0.1	23.90
MD5	0.2	17.09
RIPEMD	0.3	12.00
RIPEMD-128	0.4	9.73
SHA-1	0.5	6.88
RIPEMD-160	0.6	5.69
Tiger	0.7	4.36
Snefru-128	0.8	0.75
MD2	0.9	0.53
Snefru-256	1.0	0.50

Let s_i^g be the integrity security level of t_i, and the computation overhead of the integrity service can be calculated using Equation (8), where l_i is the amount of data whose integrity must be guaranteed, and $\mu^g(s_i^g)$ is a function used to map a security level to its corresponding hash function's performance. The security overhead model for integrity is depicted in Figure 2 (b).

$$c_i^g(s_i^g) = l_i / \mu^g(s_i^g). \quad (8)$$

- **Authentication Overhead**

Tasks must be submitted from authenticated users and, thus, authentication services are deployed to authenticate users who wish to access the Grid [8]. Table 3 enlists three authentication techniques: weak authentication using HMAC-MD5; acceptable authentication using HMAC-SHA-1, and fair authentication using CBC-MAC-AES. Each authentication technique is assigned a security level s_i^a in accordance with the performance. Thus, authentication overhead $c_i^a(s_i^a)$ is a function of security level s_i^a. The security overhead model for authentication is shown in Figure 2(c).

Table 3. Four Authentication Methods

Authentication Methods	s_i^a : Security Level	$c_i^a(s_i^a)$: Computation Time (ms)
HMAC-MD5	0.3	90
HMAC-SHA-1	0.6	148
CBC-MAC-AES	0.9	163

- **Modeling Security Overheads**

Now we can derive security overhead, which is the sum of the three items above. Suppose task t_i requires q security services, which are provided in sequential order. Let s_i^j and $c_i^j(s_i^j)$ be the security level and overhead of the jth security service, the security overhead c_i experienced by t_i, can be computed using Equation (9). In particular, the security overhead of t_i with security requirements for the three services above is modelled by Equation (10).

$$c_i = \sum_{j=1}^{q} c_i^j(s_i^j), \text{ where } s_i^j \in S_i^j. \quad (9)$$

$$c_i = \sum_{j \in \{a,e,g\}} c_i^j(s_i^j), \text{ where } s_i^j \in S_i^j. \quad (10)$$

It is to be noted that $c_i^e(s_i^e)$, $c_i^g(s_i^g)$, and $c_i^a(s_i^a)$ in Equation (10) are derived from Equations (7)-(8) and Table 2. In section 5, Equation (10) will be used to calculated the earliest start times and minimal security overhead. (See Equations 11 and Equation12).

4. The SAREG Scheduling Algorithm

The *SAREG* algorithm is outlined in Figure 3. The goal of the algorithm is to deliver high quality of security under two conditions: (1) the security level promotion will not miss its deadline; and (2) the security level promotion will not result in any accepted subsequent task to be failed. To achieve the goal, *SAREG* strives to maximize security level (measured by Equation 5) of each accepted job (see Step 21) while maintaining reasonably high guarantee ratios (see Step 12).

1. **for** each task t_k of job J_i on site M_j **do**
2. Use **Equation (11)** to computer $\sigma_{i,k}^j$, the earliest start time of task J_i on site M_j;
3. Use **Equation (12)** to obtain the minimal security overhead $c_{i,k}^{min}$ of task t_k;
4. **if** $\sigma_{i,k}^j + e_{i,k} + c_{i,k}^{min} \le d_i$ **then** (See **Property 1**)
5. Sort the security service weights in a decreasing order of their values, e.g., $w_i^{v_1} < w_i^{v_2} < w_i^{v_3}$, where $v_l \in \{a,e,g\}, 1 \le l \le 3$;
6. **for** each security service $v_l \in \{a,e,g\}, 1 \le l \le 3,$ **do**
7. $s_{i,k}^{v_l} = \min\{S_i^{v_l}\}$ /* Initialize the security value of security service v_l */
8. **for** each security service $v_l \in \{a,e,g\}, 1 \le l \le 3,$ **do**
9. **while** $s_{i,k}^{v_l} < \max\{S_i^{v_l}\}$ **do**
10. increase security level $s_{i,k}^{v_l}$;
11. Use **Equation (10)** to calculate the security overhead of t_k on M_j;
12. **if** $\sigma_{i,k}^j + e_{i,k} + \sum_{b \in \{a,e,g\}} c_{i,k}^b(s_{i,k}^b) > d_i$ (based on **Property 1**) **then**
13. decrease security level $s_{i,k}^{v_l}$; break;
14. **end while**
15. **end for**
16. $SL(s_{i,k}) = \sum_{b \in \{a,e,g\}} w_i^b s_{i,k}^b$ /* Obtain the security level of t_k on M_j using **Equation (2)** */
17. **else** Migrate t_k to another site M_r, subjecting to $\min_{1 \le r \le n, r \ne j} \left\{ \sigma_{i,k}^{j,r} + e_{i,k} + \sum_{b \in \{a,e,g\}} c_{i,k}^b(s_{i,k}^b) + \frac{\delta_i^r}{B_{r,j}} \right\} \le d_i$
18. **end for**
19. **if Property 2** is satisfied **then** /* All the tasks in J_i can be finished before d_i */
20. $y_{i,h} \leftarrow 1$, where $h = j$ or r; /* Accept job J_i */
21. /* Optimize quality of security, see **Equation (4)** */
Find site M_k for J_i, maximize $SL(s_i) = \sum_{k=1}^{p_i} \sum_{b \in \{a,e,g\}} w_i^b s_{i,k}^b$;
22. dispatch job J_i to M_k according to the schedule generated above;
23. **else** $y_{i,h} \leftarrow 0$; /* Reject J_i, since no feasible schedule is available */

Figure 3. The SAREG scheduling algorithm.

Before optimizing the security level of each task of job J_i on M_j, *SAREG* attempts to meet the real-time requirement of J_i. This can be accomplished by calculating the earliest start time (use Equation 11) and the minimal security overhead of J_i (use Equation 12) in Steps 2 and 3, followed by checking if all the tasks of J_i can be completed before the deadline d_i (see Step 4). If the deadline cannot be met by M_j, J_i is rejected by Step 23.

The security level of each task in J_i on M_j is optimized in the following way. Recall that the security service weights used in Equations (2), (4), and (6) reflect the importance of the three security services, directly indicating that it is desirable to give higher priorities to security services with higher weights (see Step 5). In other words, enhancing security levels of more important services tends to yield a maximized security level of the task on M_j.

In the case of a particular security service $v_l \in \{a, e, g\}$, Step 10 escalates the security level $s_{i,k}^{v_l}$ while satisfying the following timing constraints (see Step 12). Step 21 is able to maximize the security level of all the tasks in J_i by identifying a site M_h that provides the maximal security level and dispatching J_i to M_h (see Step 22).

The time complexity of the *SAREG* scheduling algorithm is given as follows.

Theorem 1. The time complexity of *SAREG* is *O(knm)*, where *n* is the number of sites in a Grid, *m* is the number of tasks running on a site, and *k* is the number of possible security level ranks for a particular security service v_l $(v_l \in \{a, e, g\}, 1 \le l \le 3)$.

Proof. The time complexity of finding the earliest start time for the task on a site is *O(m)* (Step 2). To obtain the minimal security overhead c_i^{min} of the task; the time complexity is a constant *O(1)* (Step 3). Sorting the security service weights in a decreasing order (Step 5) takes a constant time *O(1)* since we only have 3 security services. To increase the task's three security levels to their possible maximal ranks under the constraints (Step 12), the worst case time complexity is *O(3km)* (Step 8 ~ Step 15). To find site M_h on which the security level of task is optimized (Step 20 ~ Step 22), the time complexity is *O(n)*. Thus, the time complexity of the *SAREG* algorithm is as follows: O(*n*)(*O(m) + O(1) + O(1) + O(3km)) + O(n) = O(knm)*.

Since *n, m* and *k* cannot be very big numbers in practice, the time complexity of *SAREG* should be low based on the expression above. This time complexity indicates that the execution time of *SAREG* is a small value compared with task execution times (e.g., the real world trace used in our simulations shows that the average job execution time is 8031 Sec.). Thus, the CPU overhead of executing *SAREG-EDF* is ignored in our experiments.

5. Performance Evaluation

In the previous Section we proposed the SAREG scheduling algorithm, which integrates security requirements into scheduling for real-time applications on Grids. Now we are in a position to evaluate the effectiveness of SAREG by conducting extensive simulations based on a real world trace from San Diego Supercomputer Center (SDSC SP2 log). The real trace was sampled on a 128-node (66MHz) IBM SP2 from May 1998 through April 2000. To simplify our experiments, we utilized the first three months data with 6400 parallel jobs in simulation.

In purpose of revealing the strength of SAREG, we compared it against two well-know scheduling algorithms, namely, *Min-Min* and *Sufferage* [25] in addition to a traditional real-time scheduling algorithm - the Earliest Deadline First algorithm (*EDF)*. Min-Min and Sufferage are non-preemptive task scheduling algorithms, which were designed to schedule a stream of independent tasks onto a heterogeneous distributed computing system such as a Grid. Note that Min-Min and Sufferage are representative dynamic scheduling algorithms for Grid environments, and they were successfully applied in real world distributed resources management systems such as SmartNet [11]. For the sake of simplicity, throughout this section *Sufferage* is referred to as *SUFFER*.

To emphasize the non-security-aware characteristic of *EDF* algorithm, we refer to the EDF algorithm as *NS-EDF* (non-security-aware *EDF*) in this paper. Although the *NS-EDF* algorithm, a variation of *EDF*, is able to schedule real-time jobs with security requirements, it makes no effort to optimize quality of security. Rather, it randomly selects a security level for each task in a real-time job. The three baseline scheduling algorithms are briefly described below.

(1) *MINMIN*: For each submitted real-time job, a Grid site that offers the earliest completion time is tagged. Among all the mapped tasks, the one that has the minimum earliest completion time is chosen and then allocate to the tagged site. The MINMIN scheduler randomly selects security levels of security services required by tasks of a real-time job.

(2) *SUFFER*: Allocating a site to a submitted job that would "suffer" most in terms of completion time if that site is not allocated to it. Again, the SUFFER scheduler randomly chooses security levels for security requirements posed by an arriving job.

(3) *NS-EDF*: Tasks with the earliest deadlines are always executed first. We modified the traditional *EDF* algorithm in a way that it can randomly picks values within the security level ranges of services required by tasks. The following expression is held in the NS-EDF algorithm: $\forall 1 \le k \le p_i, b \in \{a, e, g\} : s_{i,k}^b = \text{random}\{S_i^b\}$.

The ultimate goal of comparing SAREG against MINMIN and SUFFER is to demonstrate schedulability performance improvements over existing scheduling algorithms in a real-time computing environment, whereas the purpose of comparing *SAREG* with *NS-EDF* is to show security performance benefits gained by employing SAREG in a Grid environment. This section is organized as follows. Section 5.1 describes our simulator and important system parameters. Section 5.2 is to examine the performance improvements of SAREG over the three baseline algorithms. In Section 5.3 we investigate the performance impacts of the number of computing nodes in a simulated four-site Grid. Section 5.4 addresses the performance sensitivity of the SAREG algorithm to CPU capacities of the nodes in a Grid. We evaluate in Section 5.5 the scalability (measured as Grid size) of the proposed SAREG algorithm. Last but not least, Section 5.6 demonstrates that SAREG delivers good performance in terms of conventional performance metrics, including the mean slowdown and mean response time.

Table 4. Characteristics of System Parameters.

Parameter	Value (Fixed) - (Varied)
CPU Speed	(2) – (4, 8, 16)
β (Deadline Base, or Laxity)	(50 second) – (200, 400, 800 second)
Network bandwidth	5 MB/Second
Number of sites	(4) – (8, 16, 32)
Number of nodes	(184)- (256, 320,384)
Mean size of data to be secured	50KB for short jobs, 500KB for medium jobs, 1MB for long jobs
Mean size of input data	100MB for short jobs, 500MB for medium jobs, 1TB for large jobs
Mean size of application code	500KB for short jobs, 5MB for medium jobs, 50MB for large jobs
Required security services	Encryption, Integrity and Authentication
Weights of security services	Authentication=0.2; Encryption=0.5; Integrity=0.3

5.1. Simulator and Simulation Parameters

Before presenting the empirical results in detail, we present the simulation model as follows. A competitive advantage of conducting simulation experiments is that performance evaluation on a Grid can be accomplished without additional hardware cost. The Grid simulator was designed and implemented based on the model and the algorithm described in the previous sections.

Table 4 summarizes the key configuration parameters of the simulated Grid used in our experiments. The parameters of nodes in Grid are chosen to resemble real-world workstations like IBM SP2 nodes.

We modified a real world trace[1] by adding randomly generated deadlines for all tasks in the trace. The assignment of deadlines is controlled by a deadline base, or laxity, denoted as β, which sets an upper bound on tasks' slack times. We use Equation (13) to generate job J_i's deadline d_i.

$$d_i = a_i + e_i + c_i^{max} + \beta, \tag{13}$$

where a_i and e_i are the arrival and execution times obtained from the real-world trace. c_i^{max} is the maximal security overhead (measured in ms), which is computed by Equation (14).

$$c_i^{\max} = \sum_{j\in\{a,e,g\}} c_i^j\left(\max\left\{S_i^j\right\}\right), \tag{14}$$

where $c_i^j\left(\max\left\{S_i^j\right\}\right)$ represents the overhead of the jth security service for J_i when the corresponding maximal requirement is fulfilled.

"Job number", "submit time", "execution time" and "number of requested processors" of jobs submitted to the Grid are taken directly from the trace. "size of input file", "size of application code", "size of output file" and "deadlines" are synthetically generated in accordance with the above model, since these parameters are not available in the trace. Security requirements are randomly generated from 0.1 to 1.0 for each security service. When a job has to be remotely executed to meet its deadline, we must consider its migration cost, which is factored in Equation 11. In order to measure the migration cost of a job, we need to estimate the network bandwidth and the amount of data to be transferred. Vazhkudai *et. al.* [40] measured the end-to-end bandwidth between two remote super-computing centres using GridFTP. It was discovered that the network bandwidth varies from 1.5 to 10.2 MB/sec. In our simulation experiments, the network bandwidth was randomly drawn from a uniform distribution with range 1.5 to 10.2 MB/sec., which can resemble practical network bandwidth in existing distributed systems. The

[1] http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_sp2.html

synthesized deadline weakens correlations between real-time requirement and other workload characteristics. However, in the experiments we can examine performance impacts of deadlines on system performance by controlling the deadlines as fundamental simulation parameters (see Section 5.2).

The performance metrics by which we evaluate system performance include:

security value: (see Equation 6).

guarantee ratio: measured as a fraction of total submitted jobs that are found to be schedulable).

overall system performance: defined as a product of security value and guarantee ratio.

mean slowdown: the slowdown of a job is the ratio of a job's response time to its service time, and mean slowdown is the average slowdown of all schedulable jobs in the Grid.

mean response time: the response time of a job is the time interval between the job's arrival and finish times, and mean response time is the average response time of all schedulable jobs in the Grid.

5.2. Overall Performance Comparisons

The goal of this experiment is two fold: (1) to compare the proposed SAREG algorithm against the three baseline schemes, and (2) to understand the sensitivity of SAREG to parameter β, or deadline base (Laxity). To stress the evaluation, we assume that each job arrived in the Grid requires the three security services. Without loss of generality, it is assumed that time spent handling page faults is factored in jobs' execution time.

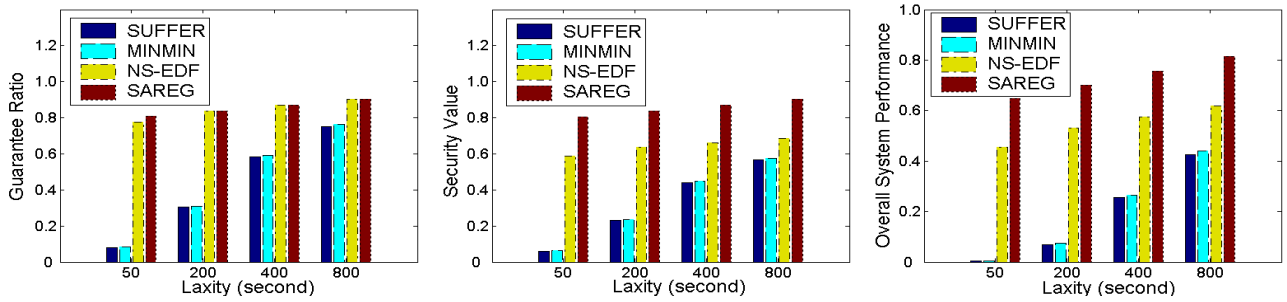
Figure 4 shows the simulation results for these four algorithms on a Grid with 4 sites (184 nodes) where the CPU power is fixed at 100MIPS. We observe from Figure 6 (a) that SAREG and NS-EDF exhibit similar performance in terms of guarantee ratio (the performance difference is less than 2%), whereas the guarantee ratios of SAREG are a lot higher than those of MINMIN and SUFFER algorithms. The reason for the performance improvements of SAREG over MINMIN and SUFFER is two fold. First, SAREG is a real-time scheduler, while MINMIN and SUFFER are non-real-time scheduling algorithms. Second, SAREG judiciously enhances the security levels of accepted jobs under the condition that the deadlines of the accepted jobs are guaranteed.

Figure 4 (a) illustrates that the guarantee ratios of four algorithms increase with the increasing value of the laxity. This is because the large deadline leads to long slack times, which in turn tend to make the deadlines more likely to be guaranteed.

Figure 4 (b) plots security values of the four alternatives when the deadline base is increased from 50 to 800 Sec. Comparing with the average execution time of all jobs in the trace, which is 8030.8 Sec., the laxity range [50, 800] is reasonable. Figure 4 (b) reveals that SAREG consistently performs better, with respect to quality of security, than all the other three approaches. When the deadlines are tight, the security values of SAREG are much higher than those of MINMIN and SUFFER. In addition, SAREG consistently outperforms NS-EDF when the laxity varies from 50 seconds to 800 seconds. This is because that SAREG can improve accepted jobs' security levels under constraints of their deadlines and resources availability, while NS-EDF makes no effort to optimize the security levels. More specifically, NS-EDF merely randomly chooses a security level within the corresponding security requirement range. Interestingly, when the deadlines become tight, the performance improvements of SAREG over the three competitor algorithms are more pronounced. The results clearly indicate that Grids can gain more performance benefits from our SAREG approach under the circumstance that real-time tasks have urgent deadlines.

The overall system performance improvements achieved by SAREG are plotted in Figure 4(c). The first observation deduced from Figure 4(c) is that the value of overall system performance increases with the laxity. This is mainly because the overall system performance is a product of security value and guarantee ratio, which become higher when the deadlines are loose due to the high laxity value.

A second observation made from figure 4(c) is that the SAREG algorithm significantly outperforms all the other



(a) Guarantee ratio **(b) Security value** **(c) Overall system performance**

Figure 4. Performance impact of deadline.

three alternatives. This can be explained by the fact that although the guarantee ratios of SAREG and NS-EDF are similar, SAREG considerably improves security values over the other algorithms, while achieving much higher guarantee ratio than MINMIN and SUFFER. This result suggests that if quality of security is the sole objective in scheduling, SAREG is more suitable for Grids than the other algorithms. By contrast, if schedulability is the only performance objective, SAREG can maintain similar guarantee ratios as those of NS-EDF, whose security performance is the second best among the four algorithms.

Last but not least, Figure 4(c) indicates that the overall performance improvement of SAREG over the other three algorithms becomes more pronounced when the deadlines are tighter, implying that more performance benefits can be obtained by SAREG for real-time applications with small slack times. This is because the SAREG approach is less sensitive to the change in deadlines than the other approaches.

5.3 Scalability

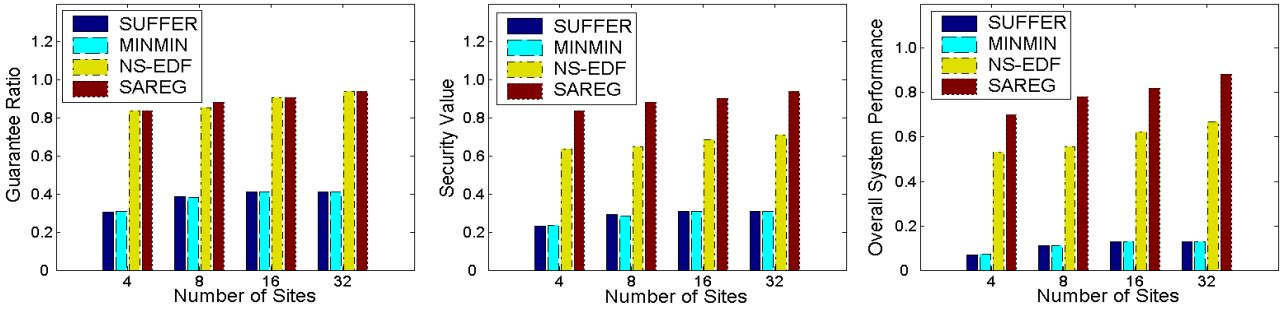
This experiment is intended to investigate the scalability of the SAREG algorithm. We scale the number of sites in the Grid from 4 to 32. Figure 5 plots the performances as functions of the number of sites in the Grid. The results show that the SAREG approach exhibits good scalability.

Figure 5 shows the improvement of SAREG in overall system performance over the other three heuristics. It is observed from Figure 5 that the amount of improvement over MINMIN and SUFFER maintains almost the same level with the increasing value of the site number. This result can be explained by the non-real-time nature of MINMIN and SUFFER, which schedules tasks that change the expected site ready time status by the least amount that any assignment could.

5.4 Sensitivities to CPU Capacity

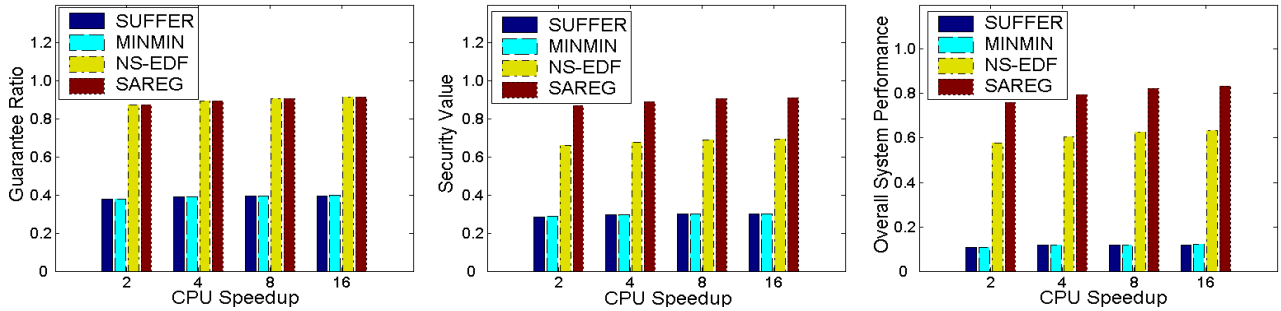
To examine performance sensitivities of the four algorithms to CPU capacity, in this set of experiments we varied the CPU capacity (measured as speedup over the baseline computational node) from 2 to 16. Specifically, the CPU speed of the IBM SP2 66MHz nodes is normalized to 1. We escalate the CPU capacity of the nodes to a normalized value of 2, 4, 8, and 16, respectively. Therefore, the execution times (including security overhead) could be 1/2, 1/4, 1/8 and 1/16 of that of original values, respectively. Also, we select a 200 seconds laxity and a four-site simulated Grid with total 184 nodes. This experiment is focused on evaluating the performance impact of CPU speedup on the four algorithms under a situation where deadlines are relatively tight and the number of nodes is less than sufficient.

The results reported in Figure 6 reveal that the SAREG



(a) Guarantee ratio **(b) Security value** **(c) Overall system performance**

Figure 5. Performance impact of number of sites.



(a) Guarantee ratio **(b) Security value** **(c) Overall system performance**

Figure 6. Performance impact of CPU Speedup.

algorithm outperforms the other three alternatives in terms of security value and overall system performance. However, the discrepancy of guarantee ratio performance between SAREG and NS-EDF is almost zero. This is because SAREG can accept the same number of submitted tasks as NS-EDF when the node's CPU speed is so fast that the security overhead is trivial and thus has little effect on the guarantee ratio performance. Figure 6 shows that MINMIN and SUFFER only slightly improve their performance when the computing capacity of the Grid is increased. The results can be explained by the following two reasons. First, the trace used in the simulation has approximate fixed job arrival rate, meaning that the decrease jobs' execution time unnecessarily improves guarantee ratio. Second, the deadlines of all submitted tasks are relatively tight. As we can see from Fig 6., the laxity has great influence on MINMIN and SUFFER in terms of the guarantee ratio.

5.5 Impact of the Number of Nodes

This subsection is focused on performance impact of the number of nodes in a four-site simulated Grid. Specifically, we evaluate the performance of the four algorithms in the cases where the total number of computation nodes in a Grid changes from 184 to 384 and all tasks have very tight deadlines (laxity=50 seconds). Each task in the trace poses requirement on how many nodes it needs. The goal is to examine the performance impact of number of total nodes in a Grid.

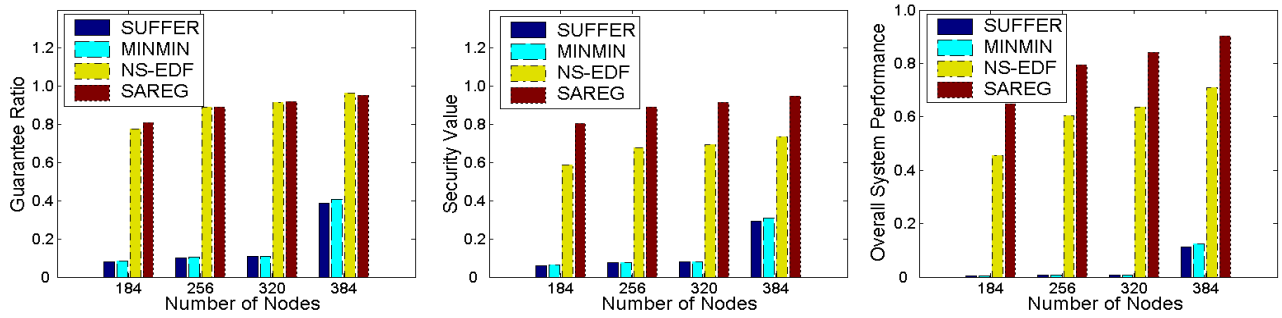
Figures 7 shows the performance impacts of the number of nodes in the Grid. We observe from the figure that the SAREG delivers better overall system performance than the other competitor algorithms under all four cases. This result is consistent with that observed from the previous experiment (see Fig. 4). Furthermore, all the four scheduling algorithms exhibit better performance when the Grid has more computation nodes. However, MINMIN and SUFFER can only marginally improve performance in guarantee ratio and security value when more computational nodes are available in the Grid. The reason is two-fold: (1) each task has extremely tight deadline and the guarantee ratios of MINMIN and SUFFER largely depend on deadlines (see Fig. 4), and (2) the number of nodes increased is very small compared with the number of total submitted jobs.

Interestingly, Figures 7 reveals that the performance improvement of *SAREG* over *NS-EDF* in terms of *GR* is not promising in the first three nodes number configurations. *NS-EDF* even outperforms *SAREG* in *GR* when the total number of nodes in the Grid is 384. The rationale behind this result is that *SAREG* always tries to promote currently arrived tasks' security levels to the maximal possible value, which in turn increase the execution time of currently scheduled tasks. Therefore, subsequent tasks could wait for a longer time to be executed and thus violate their deadlines. For *NS-EDF* this situation does not apply.

6. Summary and Future Work

In this paper, we proposed a novel scheduling algorithm, or *SAREG,* for real-time applications on computational Grids. The SAREG approach paves the way to the design of security-aware real-time scheduling algorithms for Grid computing environments. To make the SAREG scheduling algorithm practical, we presented a mathematical model in which a scheduling framework, security-sensitive real-time jobs, and security overheads are formally described. With the mathematical model in place, we can incorporate security overheads into the process of real-time scheduling. We introduced a new performance metric-security value, which was used to measure the quality of security experienced by all real-time jobs whose deadlines can be met.

To quantitatively evaluate the effectiveness of the SAREG algorithm, we conducted extensive simulations based on a real world trace from a supercomputing centre. Experimental results under a wide spectrum of workload conditions show that SAREG significantly enhances quality of security for real-time applications while



(a) Guarantee ratio **(b) Security value** **(c) Overall system performance**

Figure 7. Performance impact of number of nodes.

maintaining high guarantee ratios. Furthermore, SAREG is capable of minimizing the mean slowdown and response time under various workload characteristics. More importantly, SAREG-EDF achieves overall system performance over three existing scheduling algorithms (MIN-MIN, Sufferage, and EDF) by averages of 286.34%, 272.14%, and 33.86%, respectively.

- Future studies in this research can be performed in the following directions.
- Besides the three security services discussed, we plan to incorporate more security services into our security overhead model. Additional security services include authorization and auditing services.
- We intend to design and develop a security-aware real-time scheduling algorithm for a large scale heterogeneous Grid environment.
- To make our SAREG strategy more practical, we will extend the SAREG algorithm in a way that parallel applications with dependent tasks are considered.

Acknowledgements

This work was partially supported by a start-up research fund (103295) from the research and economic development office of the New Mexico Institute of Mining and Technology. We are grateful to five anonymous referees of this paper for their very comprehensive suggestions and comments, which greatly improved the original manuscript.

References

[1] T.F. Abdelzaher, E. M. Atkins, and K.G. Shin., "QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control," *IEEE Trans. Computers,* Vol. 49, No. 11, Nov. 2000, pp.1170-1183.

[2] Q. Ahmed and S. Vrbsky, "Maintaining security in firm real-time database systems," *Proc. 14th Ann. Computer Security Application Conf.*, 1998.

[3] F. Azzedin, M. Maheswaran, "Towards trust-aware resource management in grid computing systems," *Proc. 2nd IEEE/ACM Int'l Symp. Cluster Computing and the Grid,* May 2002.

[4] A. Bosselaers, R. Govaerts and J. Vandewalle, "Fast hashing on the Pentium," *Proc. Advances in Cryptology*, LNCS 1109, pp. 298-312, Springer-Verlag, 1996.

[5] T. D. Braun et al., "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," *Proc. Workshop on Heterogeneous Computing*, pp.15-29, Apr. 1999.

[6] E. Caron, P. K. Chouhan, and F. Desprez, "Deadline Scheduling with Priority for Client-Server Systems on the Grid," *Proc. Int'l Workshop Grid Computing*, pp. 410-414, Nov. 2004.

[7] K. Connelly and A. A. Chien, "Breaking the barriers: high performance security for high performance computing," *Proc. Workshop on New security paradigms*, Virginia, Sept. 2002.

[8] J. Deepakumara, H.M. Heys, and R. Venkatesan, "Performance comparison of message authentication code (MAC) algorithms for Internet protocol security (IPSEC)," *Proc. Newfoundland Electrical and Computer Engineering Conf.*, St. John's, Newfoundland, Nov. 2003.

[9] O. Elkeelany, M. Matalgah, K. Sheikh, M. Thaker, G. Chaudhry, D. Medhi, J. Qaddouri, "Performance analysis of IPSec protocol: encryption and authentication," *Proc. IEEE Int'l Conf. Communications*, pp. 1164-1168, New York, NY, April-May 2002.

[10] M. Eltayeb, A. Dogan, F. Ozunger, "A data scheduling algorithm for autonomous distributed real-time applications in grid computing," *Proc. Int'l Conf. Parallel Processing*, pp.388-395, Aug. 2004.

[11] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow,J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous computing environments with SmartNet," *Proc. Heterogeneous Computing Workshop*, pp.184-199, March 1998.

[12] B. George and J. Haritsa, "Secure transaction processing in firm real-time database systems," *Proc. ACM SIGMOD Conf.,* May, 1997.

[13] W. A. Halang, et al., "Measuring the performance of real-time systems," *Int'l Journal of Time-Critical Computing Systems*, 18, pp. 59-68, 2000.

[14] A. Harbitter and D. A. Menasce, "The performance of public key enabled Kerberos authentication in mobile computing applications," *Proc. ACM Conf. Computer and Comm. Security*, pp. 78-85, 2001.

[15] M. Harchol-Balter and A. Downey, "Exploiting Process Lifetime Distributions for Load Balacing," *ACM transaction on Computer Systems,* vol. 3, no. 31, 1997.

[16] L. He, S. A. Jarvis, D. P. Spooner, X. Chen, and G. R. Nudd, "Dynamic Scheduling of Parallel Jobs with QoS Demands in Muticluster and Grids," *Proc. Int'l Workshop Grid Computing*, pp.402-409, Nov. 2004.

[17] M. Humphrey, M. R. Thompson, and K. R. Jackson, "Security for Grids," *Proc. of the IEEE*, pp. 644-652, March 2005.

[18] J.-U. In, P. Avery, R. Cavanaugh, and S. Ranka, "Policy based scheduling for simple quality of service in grid computing," *Proc. Int'l Symp. Parallel and Distributed Processing*, pp. 23-32, April 2004.

[19] C. Irvine and T. Levin, "Towards a taxonomy and costing method for security services," *Proc. 15th Annual Computer Security Applications Conference*, 1999.

[20] J. Lee, B. Tierney, and W. Johnston, "Data intensive distributed computing: a medical application example," *Proc. High Performance Computing and Networking Conf.*, April 1999.

[21] K. Li, "Experimental performance evaluation of job scheduling and processor allocation algorithms for grid computing on metacomputers," *Proc. Int'l Symp. Parallel and Distributed Processing*, April 2004.

[22] S. Liden, "The Evolution of Flight Management Systems," *Proc. IEEE/AIAA 13th Digital Avionics Systems Conf.*, pp. 157-169, 1995.

[23] C. L. Liu, J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, Vol.20, No.1, pp. 46-61, 1973.

[24] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamicmatching and scheduling of a class of independent tasks onto heterogeneous comput-ing systems," Proc. *IEEE Heterogeneous Computing Workshop*, pp. 30–44, Apr. 1999.

[25] E. Nahum, S. O'Malley, H. Orman, R. Schroeppel, "Towards High Performance Cryptographic Software," *Proc. IEEE Workshop Architecture and Implementation of High Performance Communication Subsystems*, August 1995.

[26] J. Nilsson and F. Dahlgren, "Improving performance of load-store sequences for transaction processing workloads on multiprocessors," *Proc. Int'l Conference on Parallel Processing*, pp. 246-255, 21-24 Sept. 1999.

[27] X. Qin, H. Jiang, Y. Zhu, and D. R. Swanson, "Towards Load Balancing Support for I/O-Intensive Parallel Jobs in a Cluster of Workstations," *Proc. 5th IEEE Int'l Conf. on Cluster Computing*, pp.100-107, Dec. 2003.

[28] X. Qin and H. Jiang, "Improving Effective Bandwidth of Networks on Clusters using Load Balancing for Communication-Intensive Applications," *Proc. 24th IEEE Int'l Performance, Computing, and Communications Conf.,* Phoenix, Arizona, April 2005.

[29] X. Qin, "Improving Network Performance through Task Duplication for Parallel Applications on Clusters," *Proc. 24th IEEE Int'l Performance, Computing, and Communications Conference,* Phoenix, Arizona, April 2005.

[30] X. Qin, H. Jiang, D. R. Swanson, "An Efficient Fault-tolerant Scheduling Algorithm for Real-time Tasks with Precedence Constraints in Heterogeneous Systems," *Proc. 31st Int'l Conf. Parallel Processing,* pp.360-368, 2002.

[31] X. Qin and H. Jiang, "Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems," *Proc. 30th Int'l Conf. Parallel Processing,* pp.113-122, Sept. 2001.

[32] X. Qin and H. Jiang, "Data Grids: Supporting Data-Intensive Applications in Wide Area Networks," *High Performance Computing: Paradigm and Infrastructure*, edited by L. T. Yang and M. Guo, John Wiley and Sons, 2004.

[33] K. Ramamritham, J. A. Stankovic, "Dynamic task scheduling in distributed hard real-time system," *IEEE Software,* Vol. 1, No. 3, July 1984.

[34] J. Schreur, "B737 Flight Management Computer Flight Plan Trajectory Computation and Analysis," *Proc. Am. Control Conf.*, pp.3419-3429, 1995.

[35] S. H. Son, R. Zimmerman, and J. Hansson, "An adaptable security manager for real-time transactions," *Proc. 12th Euromicro Conf. Real-Time Systems*, pp. 63 – 70, June 2000.

[36] S. H. Son, R. Mukkamala, and R. David, "Integrating security and real-time requirements using covert channel capacity," *IEEE Trans. Knowledge and Data Engineering*, Vol. 12 , No. 6, pp. 865 – 879, 2000.

[37] J. A. Stankovic, M. Spuri, K. Ramamritham, G.C. buttazzo, "Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms*," Kluwer Academic Publishers*, 1998.

[38] M. D. Theys, M. Tan, N. Beck, H. J. Siegel, and M. Jurczyk, "A mathematical model and scheduling heuristic for satisfying prioritized data requests in an oversubscribed communication networks," *IEEE Trans. Parallel and Distributed Systems*, Vol. 11, No. 9, pp. 969-988, Oct. 2000.

[39] M.E. Thomadakis and J.-C. Liu, "On the efficient scheduling of non-periodic tasks in hard real-time systems," *Proc. 20th IEEE Real-Time Systems Symp.*, pp.148-151, 1999.

[40] S. Vazhkudai , J. M. Schopf , and I. Foster, "Predicting the Performance of Wide Area Data Transfers," *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, April 2002.

[41] R. Wright, D. J. Shifflett, C. E. Irvine, "Security Architecture for a Virtual Heterogeneous Machine," *Proc. 14th Ann. Computer Security Applications Conference*, 1998.

[42] M. Wu and X.-H Sun, "Memory conscious task partition and scheduling in Grid Environments," *Proc. Int'l Workshop Grid Computing*, pp.138-145, Nov. 2004.

[43] T. Xie, A. Sung, and X. Qin, "Dynamic Task Scheduling with Security Awareness in Real-Time Systems", *Proc. Int'l Symp. Parallel and Distributed Processing*, *the 4th Int'l Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems,* IEEE/ACM, April 2005.

[44] T. Xie, A. Sung, and X. Qin, "Dynamic Task Scheduling with Security Awareness in Real-Time Systems", *Proc. Int'l Symp. Parallel and Distributed Processing*, April 2005.