This paper appeared in the *Proceedings of the 34th International Conference on Parallel Processing* (ICPP 2005), pp. 5-12, Norway, June 14-17, 2005.

SAREC: A Security-Aware Scheduling Strategy for Real-Time Applications on Clusters

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

Abstract

Security requirements of security-critical real-time applications must be met in addition to satisfying timing constraints. However, conventional real-time scheduling algorithms ignore the applications' security requirements. In recognition that an increasing number of applications running on clusters demand both real-time performance and security, we investigate the problem of scheduling a set of independent real-time tasks with various security requirements. We propose a security overhead model that is capable of measuring security overheads incurred by security-critical tasks. Further, we propose a security-aware scheduling strategy, or SAREC, which integrates security requirements into scheduling for real-time applications by employing our security overhead model. To evaluate the effectiveness of SAREC, we implement a security-aware real-time scheduling algorithm (SAREC-EDF), which incorporates the earliest deadline first (EDF) scheduling algorithm into SAREC. Extensive simulation experiments show that SAREC-EDF significantly improves overall system performance over three baseline scheduling algorithms (variations of EDF) by up to 72.55%.

1. Introduction

With rapid advances in processing power, network bandwidth, and storage capacity of commodity off-the-shelf PCs in recent years, clusters have increasingly become the most cost-effective and viable platforms for scientific applications [21][22]. It becomes crucial to take advantage of cluster systems, where nodes are interconnected through high-speed networks, e.g. Myrinet or fast Ethernet, to meet the needs of highly complex scientific problems [20].

Recently there have been some efforts devoted to development of real-time applications on clusters [16] [23][24]. 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.

In addition to satisfying timing constraints in real-time applications, security is usually required in many applications [2][10]. Today there exist a growing number of systems that have real time and security considerations, because sensitive data and processing require special safeguard and protection against unauthorized access. In particular, real-time applications running on clusters require security protections to completely fulfill their security-critical needs. However, conventional real time systems, which are developed to guarantee timing constraints while possibly posing unacceptable security risks, are not adequate for real-time applications with requirements of information security and assurance.

In recognition that an increasing number of applications on clusters demand both real-time capabilities and security, we proposed a security-aware scheduling strategy, or SAREC, which is intended to integrate security requirements into real-time scheduling for applications running on clusters. SAREC can achieve high quality of security for real-time applications while meeting timing constraints imposed by these applications.

The contributions of this paper include: (1) an analysis of security and real-time performance needs of various applications running on clusters; (2) a security overhead model used to quantitatively measure overhead posted by various security services and security levels; (3) an security-aware real-time scheduling strategy; (4) definition of security and real-time performance metrics to evaluate our approach; and (5) a simulator where the SAREC-EDF algorithm is implemented and evaluated.

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.

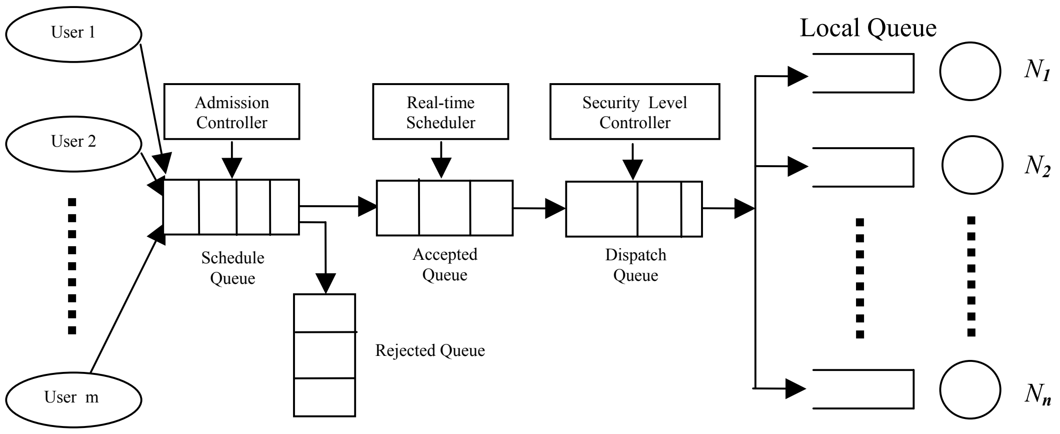


Figure 1. System model of the SAREC strategy.

2. Related work

Scheduling algorithms for clusters have been extensively studied in the past both experimentally and theoretically [29][31]. Subramani et al. incorporated a buddy scheme for contiguous node allocation into a backfilling job scheduler for clusters [29]. Vallee et al. proposed a global scheduler architecture that can dynamically change scheduling policies while applications are running on clusters [31]. 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, where various aspects of a complicated scheduling problem were addressed. In practice, real-time scheduling algorithms generally fall into two camps: static (off-line) [1] and dynamic (on-line) [7]. While many algorithms assume that real-time tasks are independent of one another [30], others schedule tasks with precedence constraints [1], which are represented by directed acyclic graphs. Conventional real-time scheduling algorithms such as Rate Monotonic (RM) algorithm [19], Earliest Deadline First (EDF) [28], and Spring scheduling algorithm [25] were successfully applied in real-time systems. However, most of existing real-time scheduling algorithms perform poorly for real-time and security-sensitive applications due to the oversight and ignorance of security requirements imposed by the applications.

Recently increasing attention has been drawn toward security-awareness in the context of clusters [3], because security has become a baseline requirement. 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 [32]. Connelly and Chien proposed an approach to protecting tightly coupled, high-performance component communication [8]. Azzedin and Maheswaran integrated the notion of "trust" into resource management of a large-scale wide-area system [4]. However, the aforementioned 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 [26]. 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]. 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 strategy, which can meet security constraints in addition to real-time requirements of tasks running on clusters. In our previous study, we proposed a dynamic security-aware scheduling algorithm for a single machine [33]. Simulation results show that the proposed algorithm can improve system performance under a wide range of workload conditions.

3. Security and Real-Time Requirements

3.1. System Model

In this study, we consider the queuing architecture of an n-node cluster in which n identical nodes are connected via a high-speed network to process soft real-time tasks submitted by *m* users. Let $N = \{N_1, N_2, ..., N_n\}$ denote the set of identical nodes. The system model, depicted in Fig. 1, is composed of a security level controller, an admission controller, and a real-time scheduler where the earliest deadline first algorithm (EDF) is applied in our experiment. The function of the admission controller is to determine if an arriving task in the schedule queue can be accepted or not, whereas the security level controller is intended to maximize the security levels of admitted tasks.

A schedule queue used to accommodate incoming real-time tasks is maintained by the admission controller. If the incoming tasks can be scheduled, the admission controller will place the tasks in the accepted queue for further processing. Otherwise, the task will be dropped into the rejected queue. The real-time scheduler processes all the accepted tasks by its scheduling policy before transmits them into the dispatch queue, where the security level controller escalates the security level of the first task 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. After being handled by the security level controller, the tasks are dispatched to one of the designated node $N_i \in N$ referred to as processing nodes for execution. The processing nodes, each of which maintains a local queue, can execute tasks in parallel.

3.2. Real-time tasks with security requirements

We consider a class of real-time systems where an application is comprised of a collection of tasks performed to accomplish an overall mission. It is assumed that tasks with soft deadlines are independent of one another. Each task requires a set of security services with various security levels specified by a user. Values of security levels are normalized to the range from 0 to 1. Note that the same security level value in different security services may have various meanings.

Suppose there is a task T_i submitted by a user, T_i is modeled as a set of rational parameters, e.g., $T_i = (a_i, e_i, f_i, d_i, l_i, S_i)$, where a_i, e_i, and f_i are the arrival, execution, and finish times, 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 [7]. Suppose T_i, requires q security services, $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, a vector of security level ranges, characterizes the security requirements of the task. S_i^j is the security level range of the jth security service required by T_i. Furthermore, the security value controller 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 security-aware scheduler has to make use of a function to measure the security benefits gained by each admitted task. In particular, the security benefit of task T_i is quantitatively modeled as a security level function denoted by SL: $S_i \rightarrow \mathfrak{R}$, where $\mathfrak{R}$ is the set of positive real numbers: $SL(s_i) = \sum_{j=1}^{q} w_i^j s_i^j$, $0 \le w_i^j \le 1$ and $\sum_{j=1}^{q} w_i^j = 1$. (1)

Note that w_i^j is the weight of the jth security service for task T_i.

Let X_i be all possible schedules for task T_i, and $x_i \in X_i$ be a scheduling decision of T_i. x_i is a feasible schedule if (1) deadline d_i can be met, e.g., $f_i \le d_i$, and (2) the security requirements are satisfied, e.g., $\min(S_i^j) \le s_i^j \le \max(S_i^j)$. Given a real-time task T_i, the security benefit of T_i is expected to be maximized by the security level controller (See Fig. 1) under the timing constraint:

$$SB(x_i) = \max_{x_i \in X_i}\{SL(s_i(x_i))\} = \max_{x_i \in X_i}\left\{\sum_{j=1}^{q} w_i^j s_i^j(x_i)\right\}, \quad (2)$$

where the security level of the jth service $s_i^j(x_i)$ is obtained under schedule x_i, and $\min(S_i^j) \le s_i^j(x_i) \le \max(S_i^j)$. $\min(S_i^j)$ and $\max(S_i^j)$ are the minimum and maximum security requirements of task T_i.

A security-aware scheduler strives to maximize the system's quality of security, or security value, defined by the sum of the security levels of admitted tasks (See Equation 1). Thus, the following security value function needs to be maximized, subjecting to certain timing and security constraints:

$$SV(x) = \max_{x \in X}\left\{\sum_{i=1}^{p} y_i SB(x_i)\right\}, \quad (3)$$

where p is the number of submitted tasks, y_i is set to 1 if task T_i is accepted, and is set to 0 otherwise. Substituting Equation (2) into (3) yields the following security value objective function. Thus, our proposed security-aware scheduling algorithm makes an effort to schedule tasks in a way to maximize Equation (4):

$$SV(x) = \max_{x \in X}\left\{\sum_{i=1}^{p}\left(y_i \max_{x_i \in X_i}\left\{\sum_{j=1}^{q} w_i^j s_i^j(x_i)\right\}\right)\right\} \quad (4)$$

4. Security Overhead Model

Since security is achieved at the cost of performance degradation, it is fundamental to quantitatively measure overheads posed by various security services [17]. To enforce security in real-time applications while making security-aware scheduling algorithms practical, in this section we proposed an effective model that is capable of measuring security overheads experienced by tasks with 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 Equations 10 and 11). Without loss of generality, we consider three security services widely deployed in real-time systems, namely, encryption, integrity, and authentication. The security overhead model (described in section 4.4) consists of the following three overhead items (section 4.1~4.3).

4.1. 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 the 3DES encryption algorithm is applied to a real-time cluster consisting of 100 MIPS machines. The time complexity of 3DES indicates an 800 bps (bit per second) encryption rate on a 100 MIPS machine [11]. Further, computation overhead caused by encryption is a linear function of the amount of data (input file size) to be protected [11]. As mentioned in Section 3.2, l_i (measured in KB) denotes the amount of data in task T_i needed to be protected. Let π_i^e (measured in milliseconds) be the CPU time spent in encrypting all data of T_i, and π_i^e is obtained by:

$$\pi_i^e = (l_i / 120 \text{ bytes})* 1.2 \text{ ms} = 10.24\, l_i \text{ ms} \quad (5)$$

Let s_i^e ($s_i^e \in [0,1.0]$) be the encryption security level. If 10% of data l_i has to be encrypted, the value s_i^e is set to 0.1. Similarly, setting the value of s_i^e to 1.0 indicates that all data must be encrypted. Given a task T_i with encryption security level s_i^e, the computation overhead for encryption is referred as c_i^e, which can be computed by Equation (6).

$$c_i^e(s_i^e) = \pi_i^e s_i^e \text{, where } s_i^e \in S_i^e \quad (6)$$

Table 1. Hash Functions Used for Integrity

Hash Functions	s_i^g Security Level	$\mu^g(s_i^g)$ 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
Snefru-256	0.9	0.50

4.2. 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 [5]. Nine commonly used hash functions and their performance (evaluated on a 90 MHz Pentium machine) are shown in Table 1. Based on their performance, each hash function is assigned a corresponding security level in the range from 0.1 to 0.9. For example, level 0.1 implies that we use MD4, which is the fastest hash function among the alternatives. Level 0.9 means that Snefru-256 is employed for integrity, and Snefru-256 is the slowest yet strongest function among the competitors.

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 (7), 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.

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

4.3. Authentication Overhead

Tasks must be submitted from authenticated users and, thus, authentication services are deployed to authenticate users who wish to access clusters [9][11][14]. Table 2 lists three authentication techniques: weak authentication using HMAC-MD5; acceptable authentication using HMAC-SHA-1, fair authentication using CBC-MAC-AES. Each authentication technique is assigned a security level s_i^a based on the performance. Thus, authentication overhead $c_i^a(s_i^a)$ is a function of security level s_i^a.

Table 2. 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

4.4. Security Overhead Model

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 (8). The security overhead of T_i with security requirements for the three services above is modeled by Equation (9).

$$c_i = \sum_{j=1}^{q} c_i^j(s_i^j) \text{, where } s_i^j \in S_i^j \quad (8)$$

$$c_i = \sum_{j \in \{a,e,g\}} c_i^j(s_i^j) \text{, where } s_i^j \in S_i^j \quad (9)$$

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 (9) are derived from Equations (6)-(7) and Table 2. In section 5, Equation (9) will be used to calculated the earliest start times and minimal security overhead. (See Equations 10 and 11).

```
1.for each task $T_i$ submitted to the schedule queue do
2.  for each node $N_j$ in the cluster do
3.    Use (10) to computer $\mathrm{es}_j(T_i)$,
4.    Use (11) to obtain $c_i^{min}$ of task $T_i$;
5.    if $\mathrm{es}_j(T_i) + e_i + c_i^{min} \le d_i$ then
6.      Sort the security service weights, e.g.,
        $w_i^{v_1} < w_i^{v_2} < w_i^{v_3}$, $v_l \in \{a, e, g\}, 1 \le l \le 3$;
7.      for each security service $v_l \in \{a, e, g\}, 1 \le l \le 3$, do
8.        $s_i^{v_l} = \min\{S_i^{v_l}\}$; /* Initialize the security value*/
9.      for each security service $v_l \in \{a, e, g\}, 1 \le l \le 3$, do
10.       while $s_i^{v_l} < \max\{S_i^{v_l}\}$ do
11.         increase security level $s_i^{v_l}$;
12.         Use (9) to calculate security overhead $c_i(N_j)$
13.(a)      if $\mathrm{es}_j(T_i) + e_i + c_i > d_i$ or
13.(b)      $\exists T_k \in N_j, d_k > d_i$: $\mathrm{es}_j(T_k) + e_i + c_i(N_j) > d_k$ then
14.           decrease security level $s_i^{v_l}$; break;
15.       end while
16.     end for
17.     $SL_i^j \leftarrow SL(s_i)$; /* Obtain the security level */
18.   else $SL_i^j \leftarrow 0$; /* Set the security level to 0 */
19. end for
20. if $\exists N_j \in N : SL_i^j > 0$ then
21.   $y_i \leftarrow 1$; /* Accept task $T_i$ */
22.   /* Optimize quality of security*/
      Find node $N_k$ for $T_i$, subject to: $SL_i^k = \max_{1 \le j \le n}\{SL_i^j\}$;
23.   dispatch task $T_i$ to $N_k$ based on the above schedule;
24. else $y_i \leftarrow 0$; /* Reject $T_i$ */
25. end for
26.end for
```

Figure 2. The SAREC-EDF

5. The SAREC-EDF Algorithm

Now we are in a position to evaluate the effectiveness of SAREC by developing a novel security-aware real-time scheduling algorithm, or SAREC-EDF, which incorporates the earliest deadline first (EDF) scheduling algorithm into the SAREC strategy. The schedule of a task is feasible if the task is completed before its deadline. Hence, a task has a feasible schedule on a cluster if there exists at least one node, where a valid schedule is available for the task. More formally, this fact can be express by the following property. The earliest start time $\mathrm{es}_j(T_i)$ can be computed by Equation (10).

$$\mathrm{es}_j(T_i) = r_j + \sum_{T_k \in N_j, d_k \le d_i} \left(e_k + \sum_{l \in \{a,e,g\}} c_k^l(s_k^l) \right), \tag{10}$$

where r_j represents the remaining overall execution time of a task currently running on the jth node, and $e_k + \sum_{l \in \{a,e,g\}} c_k^l(s_k^l)$ is the overall execution time of task T_k whose deadline is earlier than that of T_i. Thus, the earliest start time of T_k is a sum of the remaining overall execution time of the running task and the overall execution times of the tasks with earlier deadlines. The minimal security overhead c_i^{min} of T_i can be efficiently calculated by the following equation.

$$c_i^{min} = \sum_{j \in \{a,e,g\}} c_i^j\left(\min\{S_i^j\}\right), \tag{11}$$

where $c_i^j\left(\min\{S_i^j\}\right)$ denotes the overhead of the jth security service when the minimal requirement is satisfied.

The SAREC-EDF algorithm is outlined in Figure 2. Before optimizing the security level of task T_i on N_j, SAREC-EDF attempts to meet the real-time requirement of T_i. This can be accomplished by calculating the earliest start time (use Equation 10) and the minimal security overhead of T_i (use Equation 11) in Steps 3 and 4, followed by checking if T_i can be completed before its deadline (see Step 5). If the deadline can not be met by N_j, Step 18 sets T_i' security level on N_j to 0, implying that T_i can not be allocation to node N_j. If no node can produce a feasible schedule for T_i, it is rejected by Step 24.

6. Simulation Studies

Using extensive simulation experiments based on real trace consisting of 29695 tasks, we compared SAREC-EDF against three baseline algorithms: SHMIN-EDF, SHMAX-EDF, and SHRND-EDF. These three algorithms are variations of the conventional EDF algorithm. For the sake of simplicity, throughout this section SAREC-EDF is referred to as SAREC. Similarly, the baseline algorithms are referred to as SHMIN, SHMAX, and SHRND, respectively. The baseline algorithms are described below.

(1) SHMIN: The admission controller intentionally selects the lowest security level of each security services required by an incoming task.
(2) SHMAX: The admission controller chooses the highest security level for each security requirement posed by an arriving task.
(3) SHRND: Unlike the above two baseline algorithms, SHRND randomly picks a value within the security level range of each service required by a task.

6.1. Simulator and Simulation Parameters

Table 3 summarizes the key configuration parameters of the simulated clusters used in our experiments. The parameters of nodes are chosen to resemble real-world workstations like Sun SPARC-20 and Sun Ultra 10.

We modified the traces used in [15] by adding deadlines for all tasks. The assignment of deadlines is controlled by the parameter β (We use Tbase for β in the following figures), which sets an upper bound on tasks' slack times. We use Equation (12) to generate T_i's deadline d_i.

$$d_i = a_i + e_i + c_i^{max} + \beta, \qquad (12)$$

where a_i and e_i are the arrival and execution times obtained from the traces. c_i^{max} is the maximal security overhead, which is computed by Equation (13).

$$c_i^{\max} = \sum_{j \in \{a,e,g\}} c_i^j \left(\max\left\{S_i^j\right\}\right) \qquad (13)$$

where $c_i^j\left(\max\left\{S_i^j\right\}\right)$ is the overhead of the jth security service for T_i with the maximal requirements being met.

Table 3. Characteristics of System Parameters

Parameter	Value (Fixed) - (Varied)
CPU speed	100 million instructions/second
β (Tbase)	(100 ms) – (100, 500, …, 60000ms)
Number of nodes	(64) – (8, 16, 32, 64, 96, 128, 256)
Mean size of data to be secured	50KB for short jobs, 500KB for middle jobs, 1MB for long jobs
Required security services	Encryption, Integrity and Authentication

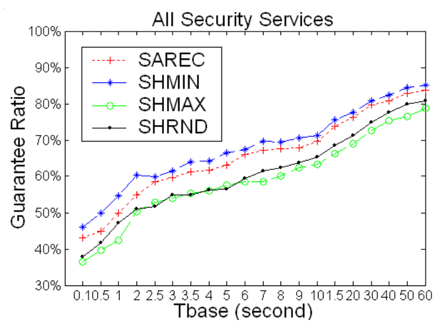
The performance metrics by which we evaluate system performance include: *security value* (see Equation 4), *guarantee ratio* (measured as a fraction of total submitted tasks that are found to be schedulable), and *overall system performance* (defined as a product of security value and guarantee ratio).

6.2. Overall Performance Comparisons

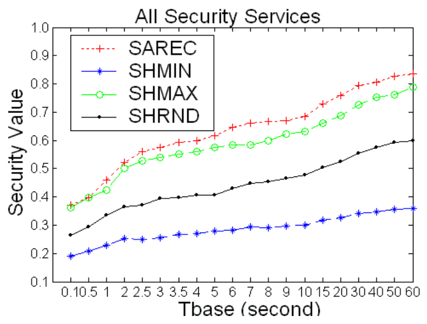
To stress the evaluation, we assume that each task arrived in the cluster requires the three security services. Figure 3 shows the simulation results for these four algorithms on a cluster with 64 nodes where the CPU power is fixed at 100MIPS. We observe from Figure 3 (a) that SAREC and SHMIN exhibit similar performance in terms of guarantee ratio (the performance difference is less than 5%), whereas the SAREC noticeably outperforms SHMAX and SHRND. We attribute the performance improvement of SAREC to the fact that SAREC judiciously boosts the security levels of accepted tasks under the condition that the deadlines of the tasks are guaranteed, thereby maintaining relatively high guarantee ratios. Unlike SAREC, SHMAX and SHRND improve quality of security at the cost of missing deadlines.

Figure 3 (a) illustrates that the guarantee ratios of four algorithms increase with the increasing value of the deadline base. This is because the large deadline base leads to long slack times, which, in turn, make the deadlines more likely to be guaranteed. Figure 3 (b) plots security values of the four algorithms when the deadline base is increased from 0.1 to 60 ms. It reveals that SAREC consistently performs better, with respect to quality of security, than SHMIN and SHRND. When the deadlines are tight, the security values of SAREC are very close to those of SHMAX. However, SAREC significantly outperforms SHMAX when the deadline base becomes large. This is because that SAREC can accept for tasks compared with SHMAX. Interestingly, when the deadlines become loose, the performance improvements of SAREC over the three competitors are more pronounced. The results clearly indicate that clusters can gain more performance benefits from SAREC under the circumstance that real-time tasks have loose deadlines.

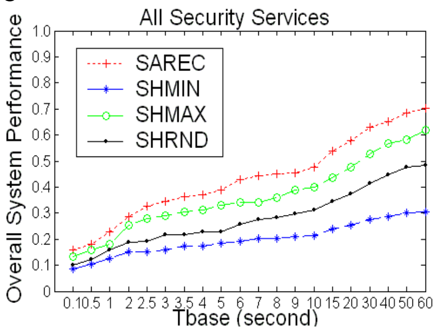
The overall system performance improvements achieved by SAREC are plotted in Figure 3(c). The first observation deduced from Figure 3(c) is that the value of overall system performance increases with the deadline base. This is because the overall system performance is a product of security value and guarantee ratio, which become higher when deadlines are loose. A second observation is that SAREC significantly outperforms the other alternatives. This can be explained by the fact that SAREC improves security values, while achieving higher guarantee ratio. Figure 3(c) indicates that the overall performance improvement achieved by SAREC becomes more pronounced when the deadlines are looser, implying that more performance benefits can be obtained for real-time tasks with large slack times.



(a) Guarantee ratio

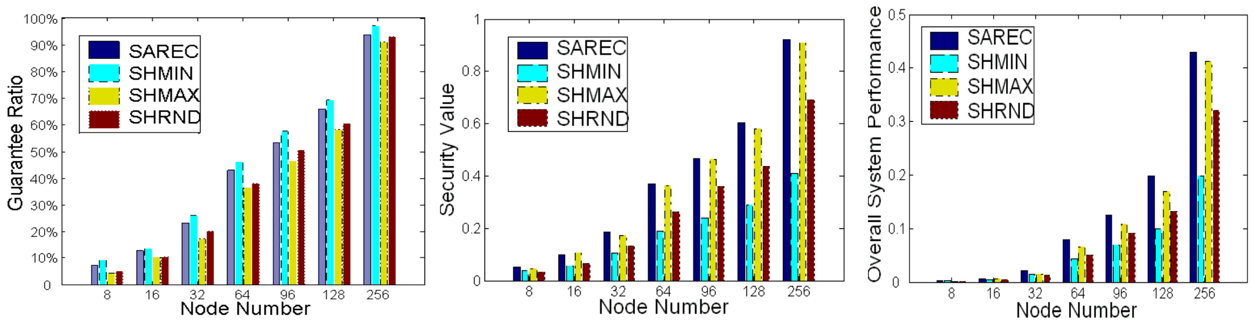


(b) Security value



(c) Overall system performance

Figure 3. Simulation performance of four scheduling algorithms.



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

Figure 4. Scalabilities of the four scheduling algorithms.

6.3. Scalability

This experiment is intended to investigate the scalability of the SAREC algorithm. We scale the number of nodes in a cluster from 8 to 256. Figure 4 plots the performances as functions of the number of nodes in the cluster. The results show that the SAREC approach exhibits good scalability.

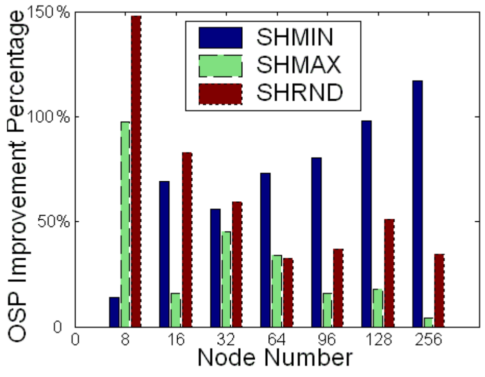


Figure 5. Impact of the number of nodes on the overall system performance improvement.

Figure 5 shows the improvement of SAREC in overall system performance over the other three heuristics. It is observed from Figure 5 that the amount of improvement over SHMIN becomes more prominent with the increasing value of node number. This result can be explained by the conservative nature of SHMIN, which merely meets the minimal security requirements for tasks accepted by the cluster. Conversely, the amount of improvement over SHMAX decreases as the number of nodes increases. This is partially because SHMAX can guarantee the maximal security requirements of more accepted tasks when more nodes are available in the cluster. It is interesting to note that the trend of the improvement over SHRND is not monotonous, because SHRND randomly decides security levels for tasks.

7. Summary and Future Work

In this paper, we presented a strategy SAREC for security-aware scheduling of real-time applications on clusters. This strategy is capable for the design of security-aware real-time scheduling algorithms like SAREC-EDF. To make security-aware scheduling algorithms practical, we also proposed a security overhead model to measure overheads of security services.

To evaluate the performance of our SAREC-EDF algorithm, we developed a trace-driven simulator and proposed two performance metrics: security value and overall system performance defined as the product of security value and guarantee ratio. Compared with three baseline algorithms, SAREC-EDF, on average, achieved improvement of 72.55%, 32.93% and 63.54%, respectively, in overall system performance. In addition, extensive experimental results show that compared with the three security-heuristic EDF algorithms above, SAREC-EDF consistently improves the overall system performance in terms of quality of security and system guarantee ratio under a wide range of workload characteristics and execution environments.

We qualitatively assigned security levels to the security services based on their respective security capacities. In our future work, we intend to investigate a quantitative way of reasonably specifying the security level of each security mechanism.

In this study we simply consider CPU time in the security overhead model. For future work, we will integrate multi-dimensional computing resources, e.g., memory, network bandwidth, and storage systems, into our model.

Acknowledgements

This work was partially supported by a start-up research fund (103295) from the research and economic development office of the New Mexico Tech and a DoD IASP Capacity Building grant.

References

[1] T.F. Abdelzaher and K.G. Shin., “Combined Task and Message Scheduling in Distributed Real-Time Systems,” *IEEE Trans. Parallel and Distributed Systems*, Vol. 10, No. 11, Nov. 1999.

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

[3] A. Apvrille and M. Pourzandi, “XML Distributed Security Policy for Clusters,” *Computers & Security Journal*, Elsevier, Vol.23, No.8, pp. 649-658, Dec. 2004.

[4] 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.

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

[6] 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.

[7] S. Cheng and Y. Huang, “Dynamic real-time scheduling for multi-processor tasks using genetic algorithm,” *Proc. Ann. Int’l Conf. Computer Software and App.*, 2004.

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

[9] 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.

[10] G. Donoho, “Building a Web Service to Provide Real-Time Stock Quotes,” *MCAD.Net*, February, 2004.

[11] 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, 2002.

[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. of the 8th 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 Trans. Computer Systems*, vol. 3, no. 31, 1997.

[16] L. He, A. Jatvis, and D. P. Spooner, “Dynamic scheduling of parallel real-time jobs by modelling spare capabilities in heterogeneous clusters,” *Proc. Int’l Conf. Cluster Computing*, pp. 2-10, Dec. 2003.

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

[18] Z. Lan and P. Deshikachar, “Performance analysis of large-scale cosmology application on three cluster systems,” *Proc. IEEE Int’l Conf. Cluster Computing*, pp. 56-63, Dec. 2003.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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. Aug. 2002.

[24] 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.

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

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

[27] 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, Nov.-Dec. 2000.

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

[29] V. Subramani, V., R. Kettimuthu, S. Srinivasan, J. Johnston, and P. Sadayappan, “Selective buddy allocation for scheduling parallel jobs on clusters,” *Proc. IEEE Int’l Conf. Cluster Computing*, pp. 107 – 116, Sept. 2002.

[30] 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 Sys. Symp.*, pp.148-151, 1999.

[31] G. Vallee, C. Morin, J.-Y. Berthou, and L. Rilling, “A new approach to configurable dynamic scheduling in clusters based on single system image technologies,” *Proc. Int’l Symp. Parallel and Distributed Processing*, April 2003.

[32] R. Wright, D. J. Shifflett, C. E. Irvine, “Security Architecture for a Virtual Heterogeneous Machine,” *Proc. 14th Ann. Computer Security Applications Conf.*, 1998.

[33] 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 Sys.*, April 2005.