Scheduling Security-Critical Real-Time Applications on Clusters

Tao Xie, *Student Member*, *IEEE*, and Xiao Qin, *Member, IEEE*

Abstract— Security-critical real-time applications such as military aircraft flight control systems have mandatory security requirements in addition to stringent timing constraints. Conventional real-time scheduling algorithms, however, either disregard applications' security needs and thus expose the applications to security threats, or run applications at inferior security levels without optimizing security performance. In recognition that many 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 build a security overhead model that can be used to reasonably measure security overheads incurred by the security-critical tasks. Next, we propose a security-aware real-time heuristic strategy for clusters (SAREC), which integrates security requirements into the scheduling for real-time applications on clusters. Further, to evaluate the performance of SAREC, we incorporate the earliest deadline first (EDF) scheduling policy into SAREC to implement a novel security-aware real-time scheduling algorithm (SAEDF). Experimental results from both real-world traces and a real application show that SAEDF significantly improves security over three existing scheduling algorithms (EDF, Least Laxity First, and First Come First Serve) by up to 266.7% while achieving high schedulability.

Index Terms—Clusters, scheduling, real-time systems, security-critical applications, security overhead model.

1 Introduction

CLUSTERS have become the most cost-effective computational platforms for scientific applications [29][30]. As typical scientific simulation and computation require a large amount of compute power, it is common practice to apply cluster computing systems where nodes are interconnected through high-speed networks to meet the needs of complex scientific computing [4][28]. Meanwhile, a growing number of real-time applications have been developed and deployed in clusters [4][19][31][32]. The correctness of real-time applications depends not only on the logical computation being performed, but also on the time at which the results are produced [16]. Real-time applications can be classified into two camps: hard real-time and soft real-time applications. Hard real-time applications require a guarantee that all real-time tasks complete within specified deadlines. Soft real-time systems, on the other hand, are less restrictive and do not require the completion of all tasks within deadlines. Examples of hard real-time applications include aircraft control, radar for tracking missiles, and medical electronics. On-line transaction processing systems are examples of soft real-time applications.

Nowadays security is of critical importance for a wide range of real-time applications on clusters [4][5][6][11][27][36]. For example, in a real-time stock quote update and trading system, incoming requests from business partners and outgoing responses from an enterprise's back-end application have deadlines and security requirements, which have to be dealt with by a cluster located between the business partners and enterprise back-end applications [13]. Unfortunately, since clusters are built to execute a broad spectrum of unverified user-implemented applications from a vast number of different users, both applications and users can be sources of security threats to clusters [47]. For example, the vulnerabilities of applications can be exploited by hackers to compromise the clusters, and malicious users can access the clusters to launch denial of service attacks. Even a legitimate user may tamper with shared data or excessively consume computing cycles to disrupt services available to other cluster users [47]. On the other hand however, many existing cluster computing environments have not employed any security mechanism to counter the security threats [11]. Thus, it is mandatory to deploy security services to protect security-critical applications running on clusters. Since snooping, alteration, and spoofing are three common attacks in cluster environments, we considered three security services (authentication service, integrity service, and confidentiality service) to guard against the common threats to clusters. Snooping, an unauthorized interception of information, can be countered by confidentiality services. Alteration is an unauthorized change of information. Integrity services can be used to cope with threats of alteration. Spoofing, an impersonation of one entity by another, can be countered by authentication services [7]. With the three security services in place, users can flexibly select the security services to form an integrated security protection against a diversity of threats and attacks in a cluster environment.

Scheduling algorithms play a key role in obtaining high performance in cluster computing [41][49]. Unfortunately, conventional real-time scheduling algorithms, which were developed to mainly guarantee timing constraints while

- *T. Xie is with the Department of Computer Science,New Mexico Institute of Mining and Technology, Socorro, NM 87801, U.S.A. E-mail: xietao@nmt.edu.*
- *X. Qin is with the D Computer Science,New Mexico Institute of Mining and Technology, Socorro, NM 87801, U.S.A. E-mail: xqin@nmt.edu.*

Manuscript received 21 Feb. 2005; Revised 15 Sept. 2005; accepted 12 Jan.2006.

xxxx-xxxx/0x/$xx.00 © 2006 IEEE

possibly ignoring security requirements, are not adequate for security-critical real-time applications on clusters. In this study we focus on a way of scheduling security-critical real-time applications in a cluster environment where the aforementioned security services are employed. We propose a security-aware real-time heuristic strategy on clusters (SAREC), which integrates security requirements into real-time scheduling for applications running on clusters. To illustrate the effectiveness of SAREC, we implement a security-aware real-time scheduling algorithm (Security-Aware EDF, or SAEDF for short) by incorporating the earliest deadline first (EDF) scheduling into SAREC. It is worth noting that SAREC is a flexible security-aware strategy in the sense that (1) the fundamental idea of SAREC can be readily extended to handle other QoS parameters such as power, fault-tolerance, etc; (2) SAREC can be easily combined together with other existing real-time scheduling policies like the least-laxity-first algorithm [24] (see Section 6.8), thereby making the existing scheduling algorithms adaptable to accommodating security requirements.

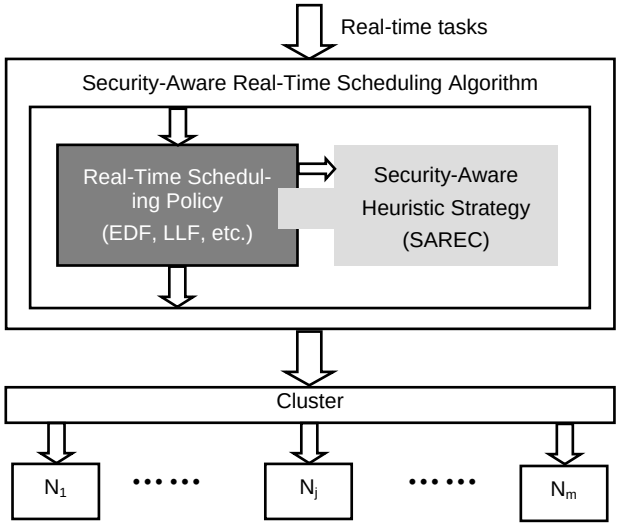


Fig. 1. Security-aware real-time scheduling framework.

Fig. 1 depicts the security-aware scheduling framework. The scheduling core implements logic and timing mechanisms for waiting, and relies on the SAREC strategy to decide quality of security for newly arrived tasks. SAREC is independent of scheduling policies, and it is implemented as a module that can perform in concert with real-time scheduling policies. In doing so, it is easy to integrate the security-aware heuristic strategy into any real-time scheduling policy.

The main contributions of this paper are: (1) an analysis of security and real-time performance needs of various applications running on clusters; (2) a security overhead model needed for quantitatively measuring overheads introduced by security services; (3) a security-aware heuristic strategy that can be integrated into existing real-time scheduling policies; (4) two new performance metrics used to evaluate the security performance of our approach; and (5) a simulated cluster where the SAEDF algorithm is implemented and evaluated.

The rest of the paper is organized as follows. Section 2 outlines related work in this area. In Section 3, we present a security-aware real-time scheduling architecture and a task model with security requirements. Section 4 proposes a security overhead model. In Section 5, we present a security-aware scheduling algorithm and investigate its properties. Performance analysis of the SAEDF algorithm is discussed in Section 6. Section7 concludes the paper with summary and future work.

2 Related Work

A large amount of work has been done to develop scheduling algorithms for clusters [39][41]. Zhang *et al.* compared the advantages of various dynamic scheduling strategies over traditional gang scheduling [49]. Subramani *et al.* incorporated a buddy scheme for contiguous node allocation into a backfilling job scheduler for clusters [39]. Vallee *et al.* proposed a global scheduler architecture that can dynamically change scheduling policies while applications are running on clusters [41]. Although these scheduling algorithms can achieve high performance for non-real-time applications, they are not suitable for real-time applications due to the lack of guarantee to finish real-time tasks to meet their deadlines.

The problem of real-time scheduling was extensively studied in the past both theoretically and experimentally. Real-time scheduling algorithms generally fall into two categories: static (off-line) [1] and dynamic (on-line) [10][21]. Many scheduling algorithms assume that real-time tasks are independent with one other [40], whereas others can schedule tasks with precedence constraints [1]. Conventional real-time scheduling algorithms like Rate Monotonic (*RM*) algorithm [25], Earliest Deadline First (EDF) [38], and Spring scheduling algorithm [33] were successfully applied in real-time systems. Most existing real-time scheduling algorithms perform poorly for applications with both time and security constraints, because they generally ignore security requirements imposed by real-time applications.

Recently increasing attention has been directed toward the issue of security in the context of clusters, because efficient and flexible security has become a baseline requirement. Apvrille and Pourzandi developed a new security policy language named distributed security policy, or DSP, for clusters [5]. Wright *et al.* proposed a security architecture for a network of computers bound together by an overlying framework that can be used to provide users a powerful virtual heterogeneous machine [42]. The language offers a precise way to customize security of clusters. Yurcik *et al.* developed tools for managing cluster security via process monitoring [46]. Connelly and Chien proposed an approach to protecting tightly-coupled, high-performance component communications [11]. Azzedin and Maheswaran applied the notion of "trust" into resource management of a large-scale wide-area system [6]. However, the security techniques mentioned above are not appropriate for real-time applications due to the lack of ability to express and handle timing constraints.

Some work was done to incorporate security into a variety of real-time applications. George and Haritsa proposed

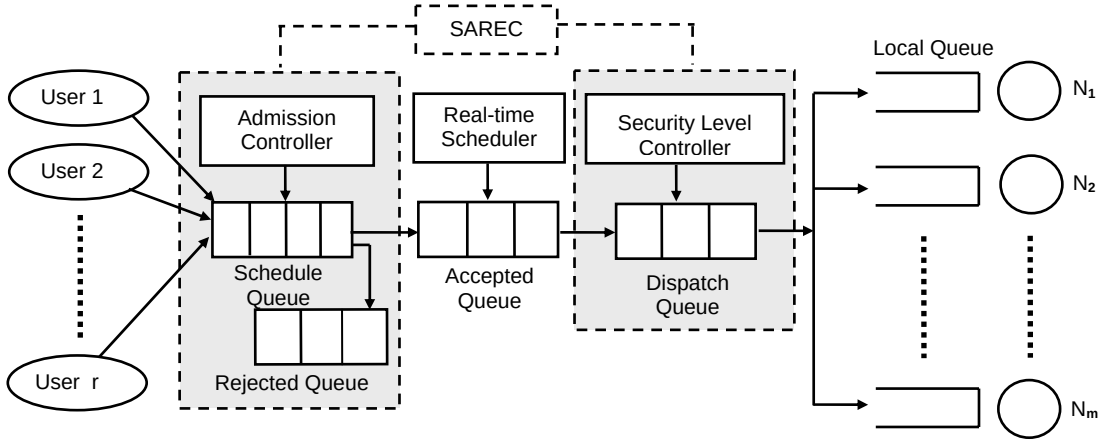


Fig. 2. Security-aware real-time scheduling architecture.

concurrency control protocols to support applications with real-time and security requirements [15]. Ahmed and Vrbsky developed a secure optimistic concurrency control protocol that can make trade-offs between security and real-time requirements [3]. Son *et al.* proposed a way of 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 our goal is to develop security-aware real-time scheduling algorithms.

Song *et al*. developed security-driven scheduling algorithms for grids [37]. Very recently we proposed a family of dynamic security-aware scheduling algorithms for single machines [45] and Grids [43]. We conducted simulations to show that the proposed algorithms can consistently improve overall system performance in terms of quality of security and system schedulability under a wide range of workload conditions.

3 Security and Real-Time Requirements

3.1 Security-Aware Scheduling Architecture

We focus in this study on an m-node cluster in which m identical nodes are connected via a high-speed network, e.g., Myrinet and Fast Ethernet, to process soft real-time tasks submitted by r users. Let $N = \{N_1, N_2, \ldots, N_m\}$ denote a set of identical computational nodes. The architecture of security-aware real-time scheduling shown in Fig. 2 encompasses the SAREC strategy and a real-time scheduler. The SAREC strategy is implemented in form of a security level controller and an admission controller. In this study we build the real-time scheduler using the EDF policy, which can be substituted by other real-time scheduling policies. The admission controller determines if an arriving task in a *schedule queue* can be accepted or not, whereas the security level controller aims at maximizing the security levels of admitted tasks.

The schedule queue maintained by the admission controller is deployed to accommodate incoming real-time tasks. If the deadline and minimal security requirements of an incoming task can be guaranteed, the admission controller will place the task in an *accepted queue* for further processing. Otherwise, the task will be dropped into a *rejected queue*. The real-time scheduler processes all the accepted tasks by its scheduling policy before the tasks are transmitted into a *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 make the first task miss its deadline; and (2) increasing security level will not make any previously accepted task miss its deadline. After being handled by the security level controller, the task is dispatched to one of the designated node $N_i \in N$ referred to as a *processing node* for execution. Each processing node maintains a *local queue*.

3.2 Real-Time Tasks with Security Requirements

We consider a class of real-time applications, each of which is composed of a collection of tasks performed to accomplish an overall mission. It is assumed in this study that tasks with soft deadlines are independent of one another. The security requirements of each task are represented by a set of security level ranges specified by a user. Values of security levels are normalized to the range from 0 to 1. For example, a task specifies security level ranges [0.25, 0.75] for the authentication service, [0.3, 0.7] for the integrity service, and [0.2, 0.8] for the confidentiality service. The higher the security levels, the more security-sensitive the task is. The same security level value in different security services has different meanings.

A task T_i submitted by a user 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 [9]. Suppose T_i requires q security services represented by a vector of security level ranges, e.g., $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$. The vector characterizes the security requirements of the task. S_i^j is the security level range of the jth security service required by T_i. The security level controller determines 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 $s_i^j \in S_i^j$, $1 \le j \le q$.

It is imperative for a security-aware scheduler to adopt a way of measuring security benefits gained by each admitted task. As such, the security benefit of task T_i is quantitatively modeled as a security level function denoted by $SL: S_i \rightarrow \Re$, where $\Re$ is the set of positive real numbers:

$$SL(s_i) = \sum_{j=1}^{q} w_i^j s_i^j \ , \ 0 \le w_i^j \le 1, \ \sum_{j=1}^{q} w_i^j = 1 \cdot \quad (1)$$

Note that w_i^j is the weight of the jth security service for task T_i. Users specify in their requests the weights to reflect relative priorities of the required security services.

X_i denotes 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 guaranteed, i.e., $f_i \le d_i$, and (2) the security requirements are met, i.e., $\min(S_i^j) \le s_i^j \le \max(S_i^j)$. 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. 2) 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 aims at maximizing 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, subject 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 is task T_i is accepted, and is set to 0 otherwise. Substituting Equation 2 into 3 yields the following security value objective function. Our proposed security-aware scheduling algorithm strives to schedule tasks in a way to maximize Equation 4:

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

4 Security Overhead Model

It is critical and fundamental to quantitatively measure overheads incurred by an array of security services, because security is achieved at the expense of performance. However, attention paid to models used to measure security overheads has been insufficient. Recently Irvine and Levin proposed a security overhead framework, which can be used for a variety of purposes [20]. Nevertheless, security overhead models for 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, we propose in this section an effective model that is capable of approximately, yet reasonably, measuring security overheads experienced by tasks with security requirements. In light of the security overhead model, schedulers are enabled to incorporate security 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, in this security overhead model we consider three security services widely deployed in clusters, namely, confidentiality, integrity, and authentication. We assume that the clusters are available, i.e., they respond tasks submitted by users. Please note that security mechanisms are not independent of one another. Rather, it is common that multiple security mechanisms are needed to form an integrated security solution, which can meet complex security demands. For example, authentication must be used in concert with message integrity. An array of primitive security services can be provided as building blocks for users to form integrated security solutions for applications. To examine the performance impact of each security service on our scheduling policies, we individually tested the three security services. This experimental strategy by no means implies that in reality security services should be separated. The security overhead model (described in section 4.4) consists of the following three items (section 4.1~4.3).

4.1 Confidentiality Overhead

Encryption mechanisms support confidentiality by enciphering real-time applications (executable files) and data such that information and resources are not made available or disclosed to unauthorized persons or processes. Suppose there are eight encryption algorithms (see Table 1) deployed in a cluster. In accordance to the cryptographic algorithms' performance, each algorithm is assigned a security level in the range from 0.08 to 1. For example, we assign security level 1 to the strongest yet slowest encryption algorithm IDEA (see Table 1). Security levels for the rest algorithms can be computed by Equation 5, where μ_i^c is the performance of the ith ($1 \le i \le 8$) encryption algorithm.

$$sl_i^c = 13.5 / \mu_i^c \ , 1 \le i \le 8. \quad (5)$$

Security levels of the algorithms are proportional to the algorithms' performance. Since computation overhead caused by encryption mainly depends on the cryptographic algorithms used and the size of data to be protected, Fig. 3a shows encryption time in seconds as a function of encryption algorithms and size of secured data measured on a 175 MHz Dec Alpha600 machine [26].

TABLE 1. CRYPTOGRAPHIC ALGORITHMS FOR CONFIDENTIALITY

Cryptographic Algorithms	sl_i^c : SL Security Level	μ_i^c :KB/ms
SEAL	0.08	168.75
RC4	0.14	96.43
Blowfish	0.36	37.5
Knufu/Khafre	0.40	33.75
RC5	0.46	29.35
Rijndael	0.64	21.09
DES	0.90	15
IDEA	1.00	13.5

Let s_i^e be the confidentiality security level of task T_i, and the computation overhead of a selected confidentiality service can be calculated using Equation 6, where l_i is the amount of data whose confidentiality must be guaranteed, and $\sigma^c(s_i^c)$ is a function used to map a security level to its

corresponding encryption method's performance.

$$c_i^c(s_i^c) = l_i / \sigma^c(s_i^c), 1 \le i \le 8. \quad (6)$$

4.2 Integrity Overhead

Integrity services ensure that no one can modify or tamper data and applications while they are executing on clusters without being detected. Integrity can be accomplished by using a variety of hash functions [8]. Seven commonly used hash functions and their performance (evaluated on a 90 MHz Pentium machine) are shown in Table 2. Based on the hash functions' performance, each function is assigned a security level in the range from 0.18 to 1.0. We assign security level 1 to the strongest yet slowest hash function Tiger (see Table 2), and security levels for the other hash functions can be calculated by Equation 7, where μ_i^g is the performance of the ith ($1 \le i \le 7$) hash function.

$$sl_i^g = 4.36 / \mu_i^g, 1 \le i \le 7. \quad (7)$$

TABLE 2. HASH FUNCTIONS FOR INTEGRITY

Hash Functions	s_i^g :Security Level	$\mu^g(s_i^g)$:KB/ms
MD4	0.18	23.90
MD5	0.26	17.09
RIPEMD	0.36	12.00
RIPEMD-128	0.45	9.73
SHA-1	0.63	6.88
RIPEMD-160	0.77	5.69
Tiger	1.00	4.36

Let s_i^g be the integrity security level of task T_i, and the overhead of the integrity service can be calculated using Equation 8, where l_i is the amount of data whose integrity must be achieved, and $\sigma^g(s_i^g)$ 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 Fig. 3b.

$$c_i^g(s_i^g) = l_i / \sigma^g(s_i^g), 1 \le i \le 7. \quad (8)$$

4.3 Authentication Overhead

It is of necessity that tasks are submitted from authenticated users and, therefore, authentication services are deployed to authenticate users who intend to access clusters [12][14][17].

Table 3 illustrates 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 in accordance with the performance. We assign security level 1 to the CBC-MAC-AES method. Security levels for the other two methods can be obtained using Equation 9, where μ_i^a is the performance of the ith ($1 \le i \le 3$) authentication method.

$$sl_i^a = \mu_i^a / 163, 1 \le i \le 3. \quad (9)$$

TABLE 3. AUTHENTICATION METHODS

Authentication Methods	sl_i^a : Security Level	μ_i^a : Computation Time (ms)
HMAC-MD5	0.55	90
HMAC-SHA-1	0.91	148
CBC-MAC-AES	1	163

Authentication overhead $c_i^a(s_i^a)$ of task T_i is a function of T_i's security level s_i^a. The security overhead model for authentication is shown in Fig. 3c.

4.4 Security Overhead Model

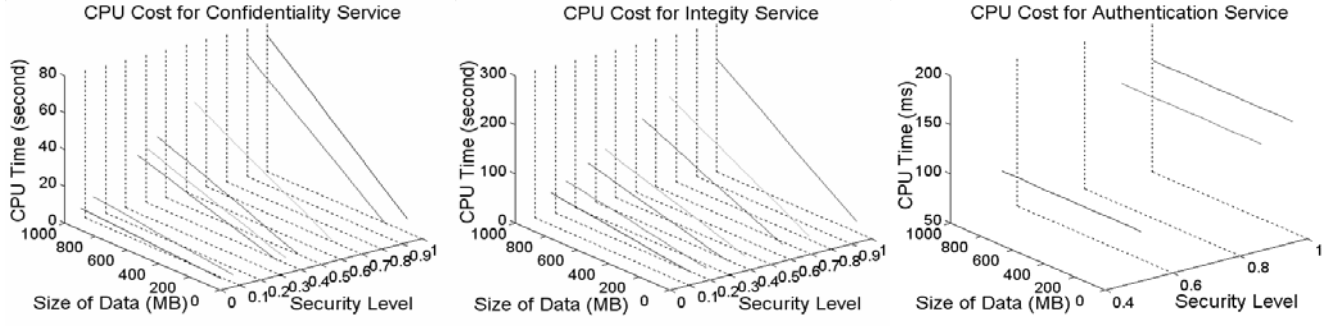
We can derive security overhead, which is the sum of the overheads imposed by all involved security services. Suppose task T_i requires q security services provided in sequential order. Let s_i^j and $c_i^j(s_i^j)$ be the security level and overhead of the jth security service, the security overhead c_i experienced by T_i, can be computed using Equation 10. In particular, the security overhead of T_i with security requirements for the three services above is measured by Equation 11.

$$c_i = \sum_{j=1}^{q} c_i^j(s_i^j), \text{ where } s_i^j \in S_i^j. \quad (10)$$

$$c_i = \sum_{j \in \{a,c,g\}} c_i^j(s_i^j), \text{where } s_i^j \in S_i^j. \quad (11)$$

Noted that $c_i^c(s_i^c)$, $c_i^g(s_i^g)$, and $c_i^a(s_i^a)$ in Equation 11 are derived from Equation 6, Equation 8 and Table 3. In the subsequent section, Equation 11 will be applied to calculated the earliest start times and minimal security overhead (See Equation 12 and Equation 13).

5 THE SAEDF ALGORITHM



(a) (b) (c)

Fig. 3. Security overhead model.

In Section 3 we proposed the SAREC strategy. Now we evaluate the effectiveness of SAREC by proposing a novel security-aware real-time scheduling algorithm, SAEDF (Security-Aware EDF), which incorporates the earliest deadline first (EDF) scheduling algorithm into the SAREC strategy.

To support the presentation of the proposed algorithm, it is necessary to introduce three properties. 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 expressed by the following property.

Property 1. If task T_i has a feasible schedule on a cluster with m nodes denoted by a set $N = \{N_1, N_2, \ldots, N_m\}$, the following inequality must be satisfied:

$$\exists N_j \in N : \mathrm{es}_j(T_i) + e_i + c_i^{min} \le d_i, \text{ under the condition stated below}$$

$$\forall T_k \in N_j, d_k > d_i : \mathrm{es}_j(T_k) + e_i + c_i^{min} \le d_k,$$

where $\mathrm{es}_j(T_i)$ is the earliest start time of task T_i on node N_2, e_i and d_i are the execution time and deadline of T_i, and c_i^{min} is the security overhead experienced by T_i when its minimal security requirements are met. The condition enforced in Property 1 indicates that the execution of T_i on N_j results in no violation of any deadlines of tasks that have been admitted to the cluster.

The earliest start time $\mathrm{es}_j(T_i)$ can be computed by Equation 12.

$$\mathrm{es}_j(T_i) = r_j + \sum_{T_k \in N_j, d_k \le d_i} \left(e_k + \sum_{l \in \{a,c,g\}} c_k^l(s_k^l) \right), \quad (12)$$

where r_j represents the remaining overall execution time of a task currently running on the jth node, and $e_k + \sum_{l \in \{a,c,g\}} c_k^l(s_k^l)$ is the overall execution time (security overhead is factored in) of task T_k whose deadline is earlier than that of T_i. Thus, the earliest start time of T_i 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 calculated by the following equation.

$$c_i^{min} = \sum_{j \in \{a,c,g\}} c_i^j \left(\min \left\{ S_i^j \right\} \right), \quad (13)$$

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

Given an arrival task T_i and a node N_j ($N_j \in N$) of the cluster, the task scheduling problem is to generate a feasible task schedule, which satisfies the following two properties.

Property 2. Task T_i meets its deadline. Thus,

1. **for** each task T_i submitted to the schedule queue **do**
2. **for** each node N_j in the cluster **do**
3. Use **Equation 12** to compute $es_j(T_i)$, the earliest start time of task T_i on node N_j;
4. Use **Equation 13** to obtain the minimal security overhead c_i^{min} of task T_i;
5. **if** $\mathrm{es}_j(T_i) + e_i + c_i^{min} \le d_i$ **and** $\exists T_k \in N_j, d_k > d_i$: $\mathrm{es}_j(T_k) + e_i + c_i(N_j) \le d_k$ (**Property 1**)
6. 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 $v_l \in \{a,c,g\}, 1 \le l \le 3$;
7. **for** each security service $v_l \in \{a,c,g\}, 1 \le l \le 3$, **do**
8. $s_i^{v_l} = \min\left\{S_i^{v_l}\right\}$; /* Initialize the security value of security service v_l */
9. **end for**
10. **for** each security service $v_l \in \{a,c,g\}, 1 \le l \le 3$, **do**
11. **while** $s_i^{v_l} < \max\{S_i^{v_l}\}$ **do**
12. Increase security level $s_i^{v_l}$;
13. Use **Equation 11** to calculate security overhead $c_i(N_j)$ of T_i on N_j;
14a. **if** $es_j(T_i)+e_i+c_i > d_i$ (**Property 2**)
14b. $\exists T_k \in N_j$, $d_k > d_i$: $es_j(T_k)+e_i+c_i(N_j) > d_k$ (**Property 1**) **then**
15. decrease security level $s_i^{v_l}$; break;
16. **end while**
17. **end for**
18. $SL_i^j \leftarrow SL(s_i)$; /* Obtain the security level of T_i on N_j using **Equation 1** */
19. **else** $SL_i^j \leftarrow 0$; /* Set the security level to 0 because T_i has no feasible schedule on N_j */
20. **end for**
21. **if** $\exists N_j \in N : SL_i^j > 0$ **then**
22. $y_i \leftarrow 1$; /* Accept task T_i */
23. /* Optimize quality of security, see **Equation 2** */
Find node N_k for T_i, subject to: $SL_i^k = \max_{1 \le j \le n}\left\{SL_i^j\right\}$;
24. dispatch task T_i to N_k according to the schedule generated above;
25. **else** $y_i \leftarrow 0$/* Reject T_i, since no feasible schedule is available */
26. **end for**

Fig. 4. The SAEDF algorithm.

$\text{es}_j(T_i) + e_i + \sum_{j\in\{a,c,g\}} c_i^j(s_i^j) \le d_i$, where $s_i^j \in S_i^j$ is the security level of the jth security service.

Property 3. The security level of an accepted task T_i on node N_j is maximized at the task's arrival time under the assumption that no more tasks arrive on N_j after this arrival time.

The SAEDF algorithm is outlined in Fig. 4. The goal of the algorithm is to deliver high quality of security while guaranteeing real-time requirements for tasks running on clusters. To achieve the goal, SAEDF strives to maximize security level (see Equation 1) of each accepted task (see Step 23) while maintaining reasonably high guarantee ratios (see Step 5).

Before optimizing the security level of task T_i on N_j, SAEDF attempts to meet the real-time requirement of T_i. This can be accomplished by calculating the earliest start time (see Equation 12) and the minimal security overhead of T_i (see Equation 13) in Steps 3 and 4. Next, Step 5 checks if the cluster can meet the timing constraints of T_i and tasks, whose deadlines are later than that of T_i. If the timing constraints can not be satisfied, Step 19 sets T_i' security level on N_j to 0, indicating that T_i can not be allocated to node N_j. In case no node the cluster can produce a feasible schedule for T_i, it is rejected by Step 25.

The security level of T_i on N_j is optimized in the following way. The security service weights used in Equations 1 and 2 reflect the importance of the three security services, indicating that it is desirable to give higher priorities to security services with higher weights (see Step 6). In other words, enhancing security levels of more important services tends to yield a maximized security level of T_i on N_j.

In case of a particular security service $v_l \in \{a, c, g\}$, Step 12 escalates the security level $s_i^{v_l}$ while satisfying the following two conditions: (1) increasing the security level will not lead to the missing deadline of T_i; and (2) the increment of the security level must not result in missing deadlines of any previously admitted task. These two conditions are respectively enforced by Steps 5 and 14. Once Step 18 have finalized an array of the optimized security levels SL_i^j $(1 \le j \le n)$, Step 23 is able to further maximize the security level of T_i by identifying a node N_k that provides the maximal security level. Finally, T_i is dispatched to N_k (see Step 24).

Now we evaluate the time complexity of SAEDF as follows.

Theorem 1. *The time complexity of SAEDF is O(knm), where m is the number of nodes in the cluster, n is the number of tasks in the local queue of a node, and k is the number of possible security level ranks for a particular security service* v_l $(v_l \in \{a,c,g\}, 1 \le l \le 3)$.

Proof. The time complexity of finding the earliest start time for task T_i on a node is $O(n)$ (Step 3). To obtain the minimal security overhead c_i^{min} of task T_i; the time complexity is a constant $O(1)$ (Step 4). Sorting the security service weights in a decreasing order (Step 6) will take a constant time $O(1)$ since we only have 3 security services. To increase T_i's three security level to their possible maximal ranks under the constraints 14a and 14b, the worst case time complexity is $O(3kn)$ (Steps 10 ~ 17). To find node N_k on which the security level of task T_i is optimized (Steps 21 ~ 23), the time complexity is $O(m)$. Thus, the time complexity of the SAEDF algorithm is as follows: $O(m)(O(n) + O(1) + O(1) + O(3kn)) + O(n) = O(knm)$. □

Since n, m and k can not be very big numbers in practice, the time complexity of SAEDF should be low based on the expression above. This time complexity indicates that the execution time of SAEDF is a small value compared with task execution times. Thus, the CPU overhead of executing SAEDF is ignored in our experiments.

In what follows we prove the correctness of the SAEDF algorithm.

Theorem 2. *The SAEDF algorithm satisfies Properties 2 and 3.*

Proof. (1) First, we prove that SAEDF satisfies Property 2. A task T_i is accepted by a cluster with m nodes denoted by $N = \{N_1, N_2, \ldots, N_m\} \Rightarrow$ There is at least one node N_j ($N_j \in N$) on which T_i has a feasible schedule $\overset{\text{Property1}}{\Rightarrow}$ The two inequalities in Property1 must hold $\overset{\text{inequality 1}}{\Rightarrow}$ task T_i can be finished before its deadline $d_i \Rightarrow$ The deadline of task T_i must be met. Thus, each accepted task meets its deadline.

(2) Second, we prove that SAEDF satisfies Property 3. We can provide a proof by contradiction. There are two cases after task T_i is accepted:

(a) Task T_i is the last element in the local queue of node N_j based on the EDF order. In this case, there is no other task in the local queue of node N_j which is behind T_i. The only constraint for increasing the security level of T_i is its deadline d_i, which is enforced by Step 14a in Fig. 4. The security level of task T_i will eventually reach a critical value SL_i^{jc1} (Steps 10 ~ 18 in Fig. 4), meaning that any further increase in security level of T_i will violate its deadline d_i. Now suppose that there is a higher security level SL_i^{jb1} ($SL_i^{jb1} > SL_i^{jc1}$) for task T_i which is an accepted task on node N_j. However, this SL_i^{jb1} definitely makes T_i violate its deadline d_i based on the conclusion drawn above because of the equality $SL_i^{jb1} > SL_i^{jc1}$. SL_i^{jb1} makes T_i miss its deadline d_i $\Rightarrow$ $\text{es}_j(T_i) + e_i + c_i > d_i \Rightarrow$ T_i cannot be accepted by node $N_j \Rightarrow$ This statement contradicts our assumption that task T_i is an accepted task on N_j. Thus, SL_i^{jc1} must be the maximal security level of T_i under this situation.

(b) Task T_i is not the last element in the local queue of node N_j based on the EDF order. Thus, there exists at least one previously accepted task to be executed after T_i is finished. The timing constraint is enforced by Step 14a. The security level of task T_i will also eventually reach a critical value SL_i^{jc2} (Steps 10 ~ 18 in Fig. 4), which means that further increase in the security level of T_i will either violate T_i's deadline or the deadlines of earlier accepted tasks. Now suppose SL_i^{jc2} is not T_i's maximal security level under this circumstance and, thus, there is a larger security level SL_i^{jb2} ($SL_i^{jb2} > SL_i^{jc2}$) for task T_i, an accepted task on node N_j under this situation. However, SL_i^{jb2} will violate either

deadline d_i or the deadlines of earlier accepted tasks because of the inequality $SL_i^{jb2} > SL_i^{jc2}$.

Case one: SL_i^{jb2} violates T_i's deadline $\Rightarrow$ $es_j(T_i) + e_i + c_i > d_i \Rightarrow T_i$ cannot be accepted on node N_j, which contradicts our assumption that task T_i is an accepted task on node N_j. Thus, SL_i^{jc2} must be the maximal security level of T_i under this situation.

Case two: SL_i^{jb2} violates the deadlines of earlier accepted tasks. Thus, $\exists T_k \in N_j$ $d_k > d_i$: $es_j(T_k) + e_k + c_k^{min}(N_j) > d_k$. The implication is that the second inequality in Property 1 does not hold. Therefore, task T_i has no feasible schedule on node N_j, meaning that T_i is not an accepted task on node N_j. This statement contradicts our assumption that T_i is an accepted task on node N_j. Consequently, SL_i^{jc2} must be the maximal security level of T_i under this situation. □

6 Experimental Results

We evaluate in this section the performance of the SAEDF algorithm using extensive simulation experiments based on real world traces consisting of 29695 tasks. A competitive advantage of conducting simulation experiments is that performance evaluation on a large-scale cluster can be accomplished without additional hardware cost. To reveal performance improvements gained by our proposed algorithm,
we
compare SAEDF with three well-known scheduling algorithms, namely, EDF (Earliest Deadline First) [38], LLF (Least Laxity First) [24], and FCFS (First Come First Serve). To make the comparisons fair, we slightly modify the three algorithms in a way that they arbitrarily pick a security level within the security level range of each service required by a task. Although these algorithms are intended to schedule real-time tasks with security requirements, they make no effort to optimize quality of security. The baseline algorithms are briefly described below.

1. EDF: The task with the earliest deadline is always executed first.
2. LLF: The task with the minimal laxity (slack time) is always executed first.
3. FCFS: Tasks will be executed in the non-decreasing order of their arrival times.

The first goal of the performance evaluation is to examine the performance improvements of SAEDF over the three competitive algorithms. Second, we will investigate the performance impacts of the security overhead model presented in Section 4 on system performance in terms of security value and guarantee ratio. Especially, we pay attention to performance impacts of security service weights on the four scheduling algorithms. Third, we study the performance sensitivity of the SAEDF algorithm to CPU capacities of the nodes in a cluster. Fourth, we evaluate the scalability of the proposed SAEDF algorithm. Fifth, we assess the performance impact of security-required data size. Sixth, we compare SALLF with LLF to demonstrate that SAREC is a general strategy, which can be incorporated into not only EDF but also other existing scheduling algorithms like LLF. Last but not least, we validate the results from the synthetic real-time tasks by running a real world real-time application with SAEDF. Some preliminary results in Sections 6.2-6.3 were presented in [44].

6.1 Simulator and Simulation Parameters

Before presenting empirical results in detail, we present the simulation model as follows. Table 4 summarizes the key configuration parameters of the simulated clusters used in our experiments. The parameters of nodes in clusters are chosen to resemble real-world workstations like Sun SPARC-20 and Sun Ultra 10.

We modified the traces used in [18][48] by adding randomly generated deadlines for all tasks in the traces, which were collected from one workstation on six different time intervals. The assignment of deadlines is controlled by the deadline base (Tbase) denoted as β, which sets an upper bound on tasks' slack times. We use Equation 14 to generate T_i's deadline d_i.

$$d_i = a_i + e_i + c_i^{max} + \beta, \tag{14}$$

where a_i and e_i are the arrival and execution times obtained from the real-world traces. c_i^{max} is the maximal security overhead (measured in ms), which is computed by Equation 15.

$$c_i^{\max} = \sum_{j \in \{a,c,g\}} c_i^j \left(\max\left\{S_i^j\right\}\right), \tag{15}$$

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

TABLE 4. CHARACTERISTICS OF SYSTEM PARAMETERS

Parameter	Value (Fixed)-(Varied)
CPU speed	(100 million instructions/second or MIPS) – (100, 200,…800)
β (Deadline Base, or Tbase)	(1000 ms) – (1000, 2000, …, 100000) ms
Number of nodes	(64) – (8, 16, 32, 64, 96, 128, 256)
Size of data to be secured (MB)	([0.05, 40], [0.5, 20000], [1, 20000]) – ({ [0.1, 400], [1, 20000], [2, 20000]}, { [0.2, 400], [2, 20000], [4, 20000]}) (mean, deviation)
Required security services	(Mixed) – (Confidentiality only, Integrity only, Authentication only)
Weight of authentication	(0.2) – (0.1, 0.3)
Weight of confidentiality	(0.5) – (0.1, 0.2, 0.3, …, 0.8)
Weight of integrity	(0.3) – (0.1, 0.2, 0.3, …, 0.8)

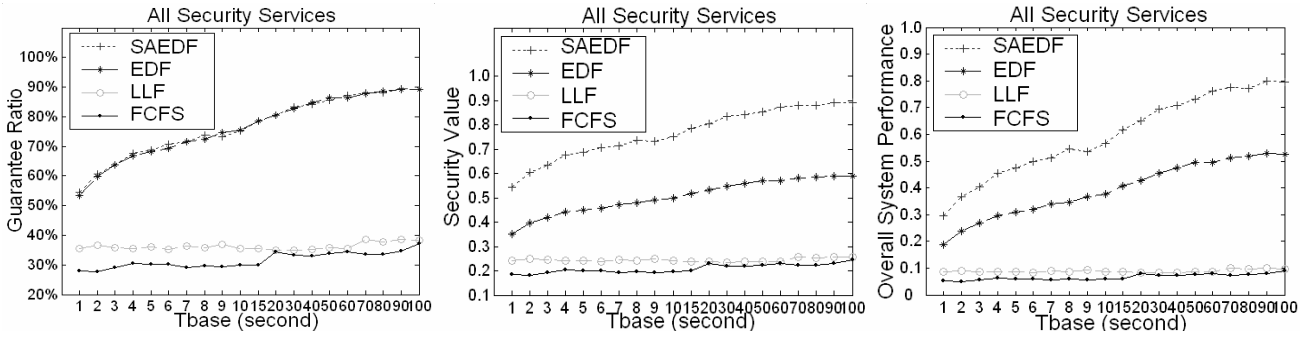


Fig. 5. Simulation performance of four scheduling algorithms.

Although CPU demands of tasks submitted to the clusters are taken directly from the existing traces, deadlines are synthetically generated in accordance with the above model. The simplification weakens correlations between real-time requirements and other workload characteristics. However, in the experiments we can examine impacts of deadlines on system performance by controlling the deadlines as fundamental simulation parameters (see Section 6.2). Similarly, each task was synthetically assigned a block of data that needs to be protected from being disclosed or tampered. The impact of security-required data size is examined in Section 6.7. The performance metrics by which we evaluate system performance include: *security value* (*SV*, see Equation 4), *guarantee ratio* (*GR*, measured as a fraction of total submitted tasks that are found to be schedulable), and *overall system performance* (*OSP*, defined as a product of normalized security value and guarantee ratio, see Equation 16).

$$OSP = GR * SV \tag{16}$$

6.2 Overall Performance Comparisons

The goal of this experiment is two fold: (1) to compare the proposed SAEDF algorithm against the three alternatives, and (2) to understand the sensitivity of SAEDF to parameter β, or deadline base (Tbase). To stress the evaluation, we assume that each task arrived in the cluster requires all of the three security services. Without loss of generality, it is assumed that no page fault occurs during the execution of each real-time task. This is because in case where a task experiences page faults, time in handling the page faults will be factored in its execution time.

Fig. 5 shows the simulation results for these four algorithms on a cluster with 64 nodes. We observe from Fig. 5a that SAEDF and EDF exhibit similar performance in terms of guarantee ratio, whereas SAEDF noticeably outperforms LLF and FCFS algorithms. Although LLF is a real-time scheduling algorithm, it does not favour short tasks as EDF does. Therefore, many subsequent short tasks are likely to miss their deadlines due to the acceptance of long tasks. FCFS has the lowest guarantee ratios, because FCFS is a non-real-time scheduling policy. It is observed that SAEDF and EDF maintain high guarantee ratios. We attribute the guarantee ratio improvement of SAEDF over LLF and FCFS to the fact that SAEDF judiciously boosts the security levels of accepted tasks under the condition that timing constraints are met. Fig. 5b plots security values of the four algorithms when the deadline base is increased from 1 to 100 seconds. Fig. 5b reveals that SAEDF consistently performs better, with respect to quality of security, than all the rest approaches. Specifically, SAEDF outperforms EDF, LLF, and FCFS in security value by averages of 43.6%, 248.9%, and 266.7%, respectively. Interestingly, when the deadlines become loose, the performance improvements of SAEDF over the three competitors are more pronounced. This is because the SAEDF approach is capable of employing slack times to improve the quality of security of accepted tasks. Therefore, the more slack time available, the higher security value can be achieved. The results clearly indicate that clusters can gain more performance benefits from the SAEDF algorithm under workload conditions where real-time tasks have loose deadlines.

Fig. 5c plots the overall system performance improvements achieved by SAEDF. An observation made from Fig. 5c is that SAEDF significantly outperforms all the other three alternatives. This can be explained by the fact that, although the guarantee ratios of SAEDF and EDF are similar, SAEDF considerably improves security values over the

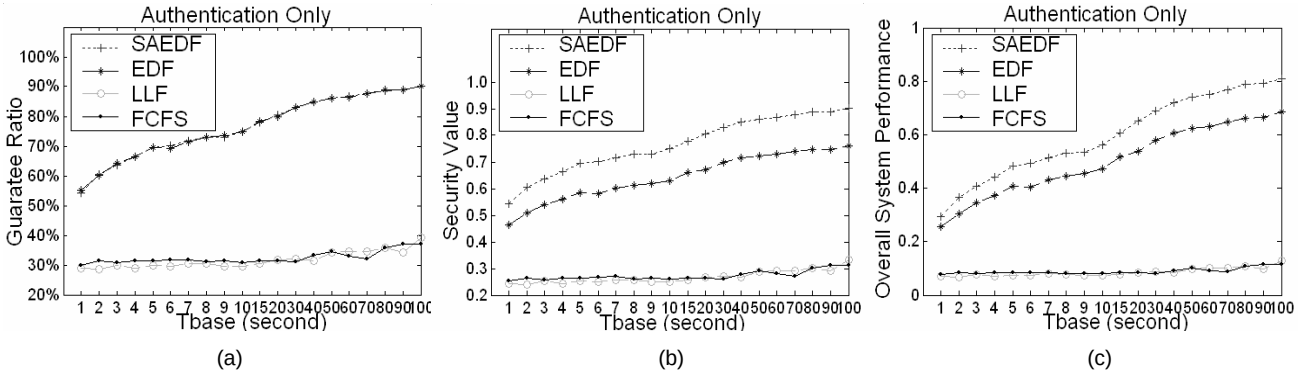


Fig. 6. Performance impact of the authentication security service.

other algorithms, while achieving higher guarantee ratio than LLF and FCFS. The result suggests that if quality of security is the sole objective in scheduling, SAEDF is more suitable for clusters than the other algorithms. In contrast, if schedulability is the only performance objective, SAEDF can maintain the same guarantee ratios as those of EDF, which is inferior to SAEDF in terms of security.

6.3 Impact of the Security Overhead Model

This subsection is focused on performance impact of the security overhead model presented in Section 4. Specifically, we evaluate the performance of the four algorithms in the cases where each task poses requirement on one of the three security services. The goal is to examine the performance impact of each security service on the scheduling policies. These experimental settings do not necessarily imply that security services should be separated. On the contrary, multiple security mechanisms in most cases are aggregated to form an integrated security solution.

Fig. 6- Fig. 8 show the performance impacts of the authentication, confidentiality, and integrity services, respectively. We observe from the figures that SAEDF delivers better overall system performance than the other competitors under a wide range of workload conditions. This result is consistent with that observed from the previous experiments (see Fig. 5), where each task requires multiple security services. Interestingly, the security improvements are more pronounced when the confidentiality or integrity service is required than when the authentication service is needed. The reason is three-fold. First, there simply exist three security levels for the authentication service in the security overhead model, and the granularity of security levels for authentication is coarser than those of the confidentiality and integrity services. Second, the authentication overhead is less than that of the confidentiality and integrity services in most cases. Thus, it is relatively easy to achieve a higher security level in the authentication service for an accepted task. Third, the confidentiality and integrity

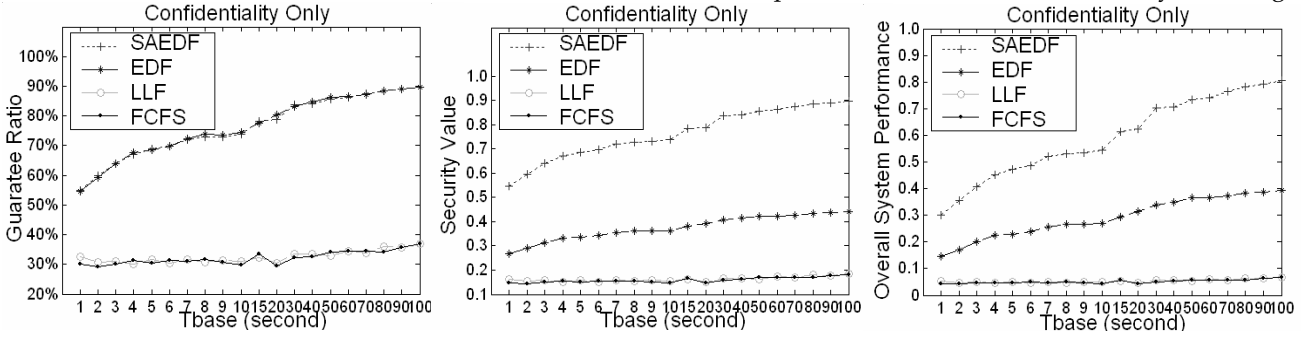


Fig. 7. Performance impact of the confidentiality security service

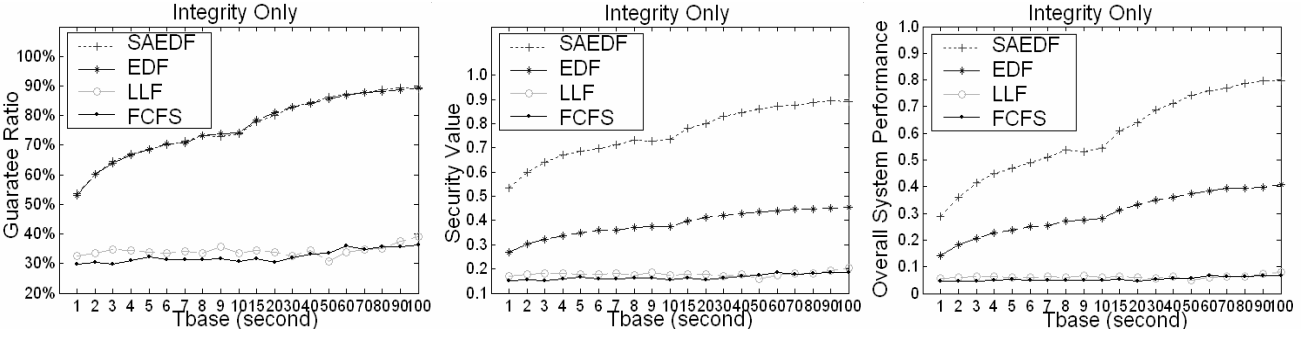


Fig. 8. Performance impact of the integrity security service

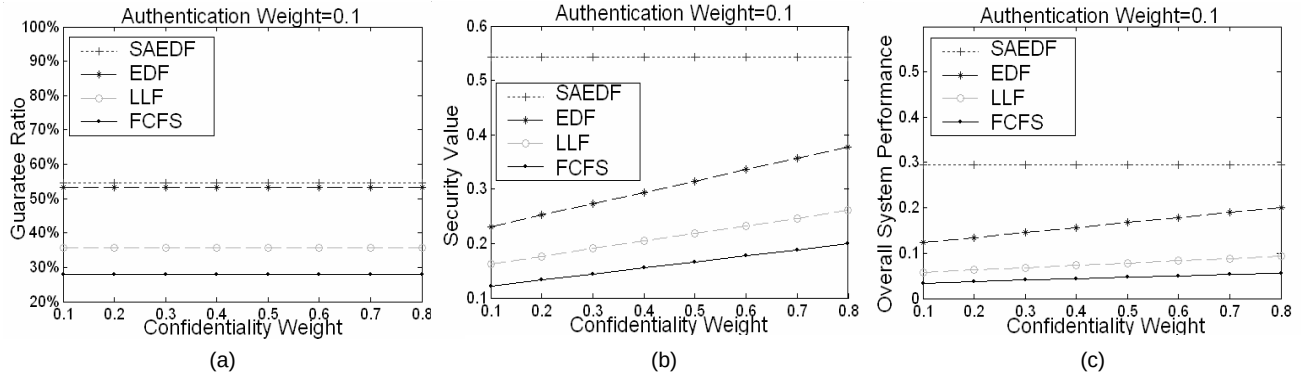
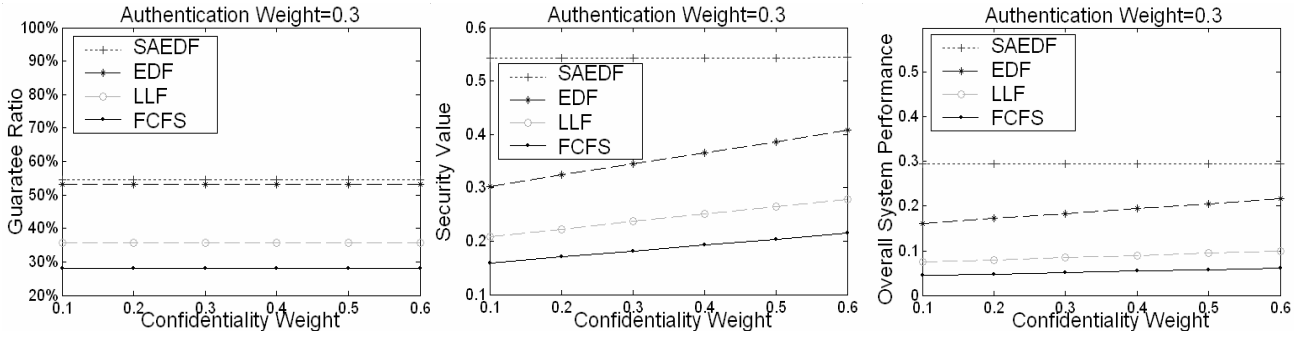


Fig. 9. Performance impact of security service weights, authentication weight = 0.1.



(a) (b) (c)

Fig. 10. Performance impact of security service weights, authentication weight = 0.3.

overheads rely on the amount of data to be protected, whereas the authentication overhead is independent of the security-required data size.

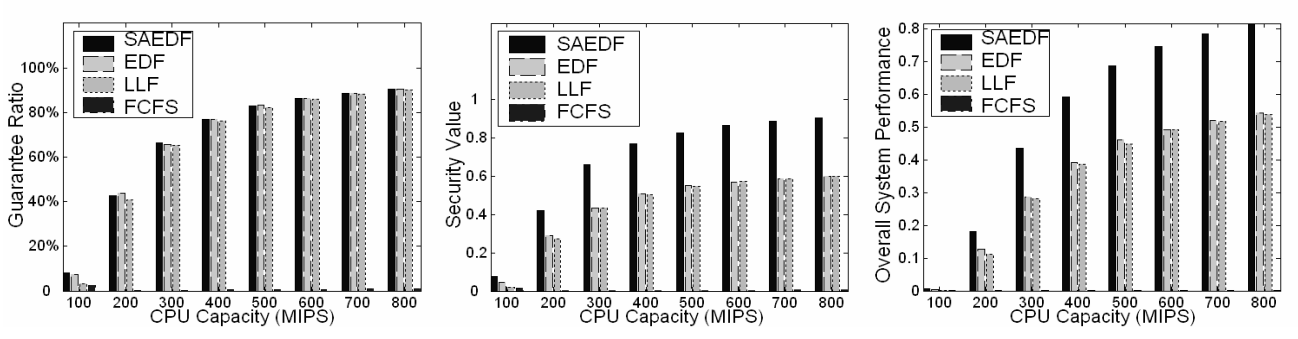
6.4 Impact of Security Service Weights

Recall that the security level model proposed in Section 3.2 is comprised of multiple security levels for a diversity of security services like confidentiality, integrity, and authentication. Each service required by a task is assigned a weight, which reflects the priority of the service. To study the impact of security service weights on performance of SAEDF, we set the authentication weight to a constant value, and varied the confidentiality and integrity weights. Specifically, Fig. 9 plots the performances of the four algorithms when the confidentiality weight is increased from 0.1 to 0.8, whereas Fig. 10 depicts the performances when the confidentiality weight varies from 0.1 to 0.6.

The first observation drawn from Fig. 9 and Fig. 10 is that for all the algorithms, the performance in guarantee ratio is independent of the security service weights. The implication of this result is that the security service weights are irrelevant to overall execution times of tasks. The second intriguing observation made from Fig. 9 and Fig. 10 is that the confidentiality and integrity weights slightly affect the security performance of SAEDF, while making considerable impact on the other three algorithms in terms of security value. This is because at the same security level, confidentiality service overhead is relatively smaller than integrity service overhead. Consequently, the overall security values of accepted tasks tend to increase when the confidentiality weight goes up. These results indicate that SAEDF can marginally improve security performance for workloads where confidentiality service is more important than the other concerns.

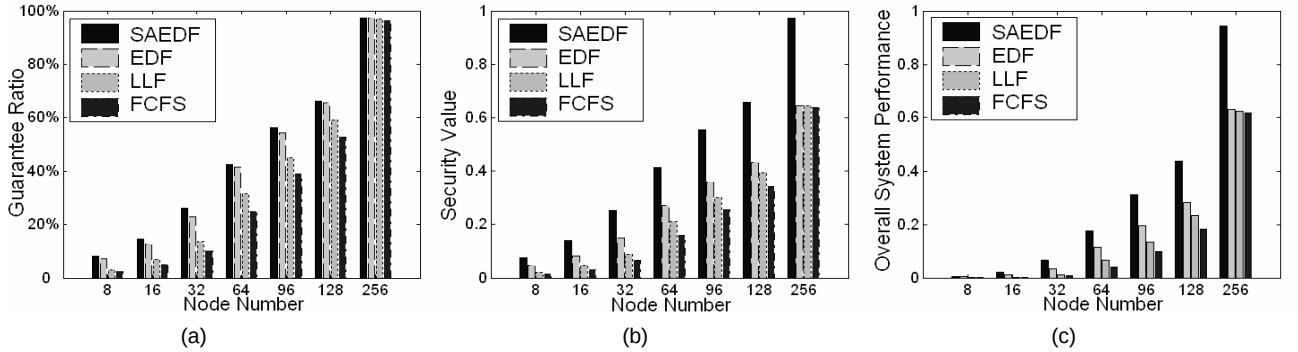
6.5 Sensitivities to CPU Capacity

To examine performance sensitivities of the four algorithms to CPU capacity, in this set of experiments we varied the



(a) (b) (c)

Fig. 11. Performance sensitivities to CPU capacity.



(a) (b) (c)

Fig. 12. Scalabilities of the four scheduling algorithms.

CPU capacity from 100 to 800 MIPS with increments of 100 MIPS.

The results reported in Fig. 11 reveal that SAEDF outperforms the other three alternatives in terms of security value and overall system performance. With respect to guarantee ratio, SAEDF exhibits a similar performance to EDF and LLF. The guarantee ratio of FCFS even decreases when CPU capacity enlarges. This is mainly because tasks with long execution times can be admitted when the CPU capacity is high and, therefore, there is a strong likelihood for more small tasks to miss their deadlines.

6.6 Scalability

This experiment is intended to investigate the scalability of the SAEDF algorithm. We scale the number of nodes in a cluster from 8 to 256. Fig. 12 plots the performances as functions of the number of nodes in the cluster. It is observed from Fig. 12 that the amount of improvement achieved by SAEDF becomes more prominent with the increasing value of the node number. This result shows that the SAEDF approach exhibits good scalability.

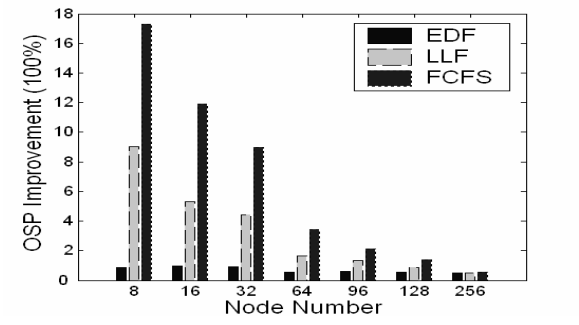


Fig. 13. Overall system performance improvement.

Fig. 13 shows the improvements of SAEDF in overall system performance over the other three policies. SAEDF outperforms the three baseline algorithms in terms of overall system performance by averages of 70.4%, 201.2%, and 625.6%, respectively.

6.7 Security-Required Data Size

In this set of experiments we evaluated the performance impact of security-required data size. We tested three configurations of data size (see Table 4). The laxity is chosen to be 1000 millisecond. Without loss of generality, we assume that the distribution of the data size is a normal distribution. The mean size of the security-required data varies from 50 KB to 4 MB and the standard deviation changes from 40 to 20000. For example, in config1, the mean size is 50KB for short tasks, 500KB for middle tasks and 1MB for long tasks. The standard deviation is set to 40 for short tasks and set to 20000 for medium and long tasks.

There are several important observations that can be drawn from Fig. 14. First, when the security-required data size increases, the guarantee ratio of SAEDF almost remains unchanged, while SAEDF's security value drops. This phenomenon reveals that SAEDF is a security-aware algorithm, which judiciously lowers accepted tasks' security levels under heavily loaded conditions in order to accommodate more tasks. Unlike SAEDF, the guarantee ratio of EDF noticeably decreases with the increasing size of security-required data. Second, Fig. 14 shows that the guarantee ratios of LLF and FCFS increase with the growing size of security-required data. This is because large tasks are more likely to be dropped due to their high security overhead caused by enlarged security-required data size. As a result, a vast majority of the small tasks submitted to the cluster can be finished before their deadlines.

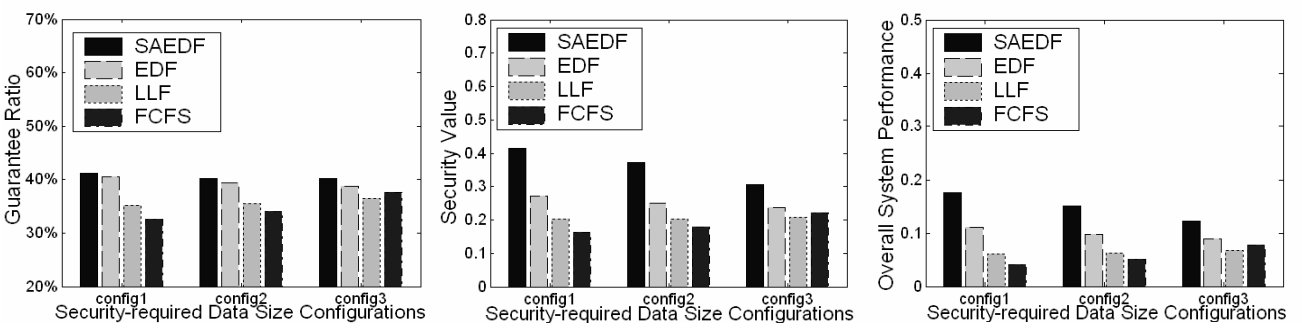


Fig. 14. Performance impact of size of data to be secured.

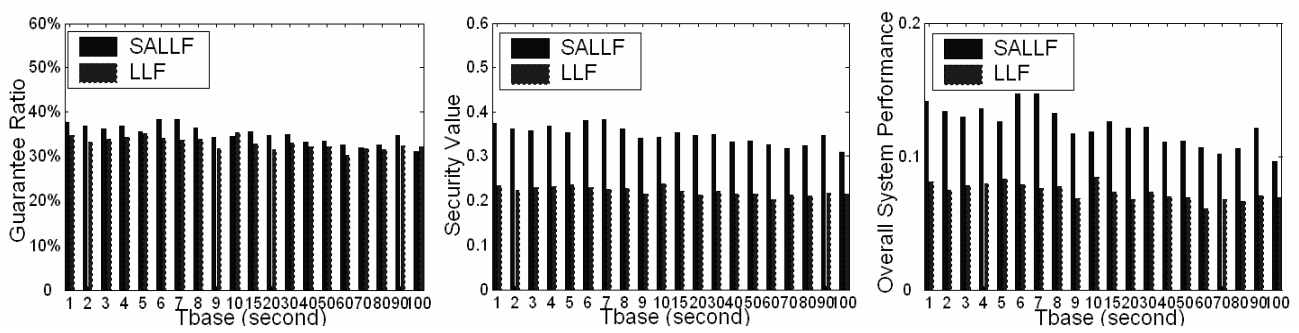


Fig. 15. Performance improvement of SALLF over LLF.

6.8 Integrate SAREC into LLF

To demonstrate that SAREC is a general security-aware strategy that can be incorporated into other existing real-time scheduling algorithms, we integrate SAREC with the least-laxity-first algorithm (LLF) [24] to construct a new algorithm called SALLF (Security-Aware LLF). Now we evaluate the performance of SALLF in this subsection.

One important observation from Fig. 15 is that SALLF outperforms LLF in all cases. Specifically, SALLF improves guarantee ratio over LLF by an average of 6.1% and outperforms LLF in terms of security value by an average of 55.8%. The rationale behind these results is that SALLF can maximize guarantee ratios by adaptively adjusting tasks' security levels, while LLF has no capability of optimizing security levels.

6.9 A Real Application – Aircraft Flight Control

To validate the results from the trace-driven simulations, we applied our SAEDF algorithm to a real world system – an automated flight control system [2]. Table 5 shows the set of parameters present for all real-time tasks, including execution time, period, and three configurations of size of data to be secured.

TABLE 5. TASK MODEL PARAMETERS FOR AUTOMATED FLIGHT CONTROL SYSTEM

Task	Exec Time (ms)	Period (sec)	Config1 (KB)	Config2 (KB)	Config3 (KB)
Guidance	100	10	300	400	500
	100	5	300	400	500
	100	1	300	400	500
Controller	80	5	200	300	500
	60	1	100	100	500
	80	1	100	100	500
	60	0.2	50	50	200
	80	0.2	50	50	200
Slow Navigation	100	10	300	400	500
	100	5	300	400	500
	100	1	300	400	500
Fast Navigation	60	5	200	300	200
	60	1	100	100	200
	60	0.2	50	50	200
Missile Control	500	10	1000	2000	1000
	500	1	500	500	1000

The automated flight control system was utilized to fly a simulated model of an F-16 fighter aircraft. Details of the automated flight control system can be found in [23][24]. In this system all the flight control tasks - including Guidance, Slow Navigation, Fast Navigation, Controller and Missile Control - need to be executed in real-time to meet their deadlines. The functions of these five tasks are summarized as follows [2]: The "Guidance" task sets the reference trajectory of the aircraft in terms of altitude and heading; the "Controller" is responsible for executing the closed-loop control functions that deal with actuator commands; the two "Navigation" tasks read sensor values distinguished by the required update frequency; and finally, the "Missile Control" task is responsible for reading radar and firing missiles. These separate tasks are mandatory to control the aircraft during flight, and they are all cyclic tasks with multiple versions.

It is assumed that (1) all tasks have known bounded execution times, (2) task arrivals are independent, (3) each task's deadline is equal to its period, and (4) tasks are non-preemptive in nature [2]. Table 5 shows that each of the five tasks has multiple versions distinguished by their period or execution time.

Since the automated flight control system is applied in a military battle field, where security requirements such as data confidentiality are mandatory, we synthetically choose security-required data sizes during the execution of each version of the five tasks. Each version of a task requires confidentiality, integrity, and authentication services. The security overhead for each task instance, which is computed using Equation 11, largely depends on the security-required data size. To evaluate the performance of our SA-EDF algorithm under various scenarios, we constructed three configurations of the security-required data size for each version of the tasks (See columns 4, 5 and 6 in Table 5). In config1, we choose a relatively low security level for each task instance. Config2 represents a medium security level, whereas config3 reflects a relatively high security level to each task version. The experimental parameters for the automated light control system are summarized in Table 6.

TABLE 6. EXPERIMENTAL PARAMETERS FOR AUTOMATED FLIGHT CONTROL SYSTEM

Parameter	Value
CPU speed	1000 MIPS
Number of nodes	128
Required security services	Confidentiality, Integrity and Authentication
Weight of authentication	0.2
Weight of confidentiality	0.5
Weight of integrity	0.3
Number of F-16 aircraft	64, 128
Sample period	600 Second

In this experiment the arrival times, deadlines, and execution times of task instances are based on the real application. The arrival time of a task instance T_i can be derived from T_i's period, and the deadline of T_i is equal to T_i's period. All five tasks start to submit their task instances to the system at the same time, e.g., start time T_s. Each task randomly selects one of its versions and submits it to the system. The rationale behind the random task version submission is that available system resources are dynamic and unpredictable. For example, when some nodes in the system fail at an unknown time, an inferior version of a task will be executed. We sampled task instances that were submitted to the system within 600 seconds since T_s, because the system behaviour was already manifest after this period of time. In this experiment, we sampled 1293 task

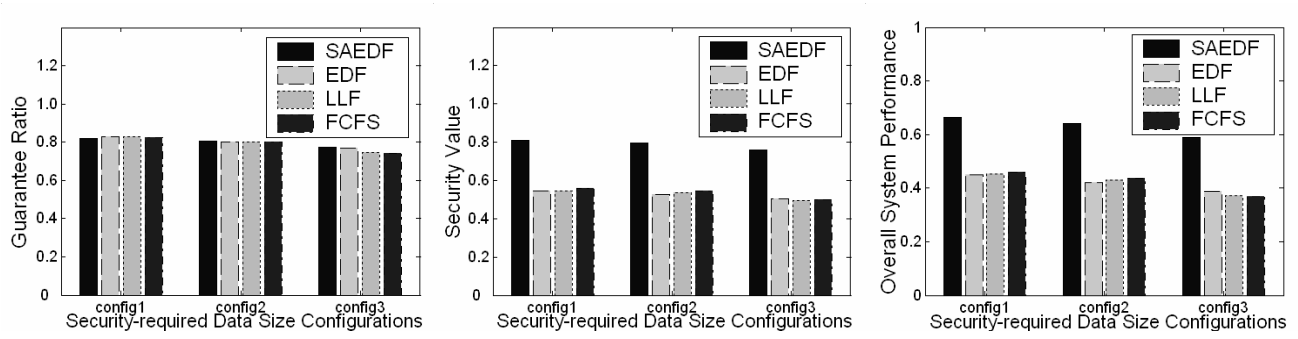


Fig. 16. 128 nodes control 128 F-16 aircrafts.

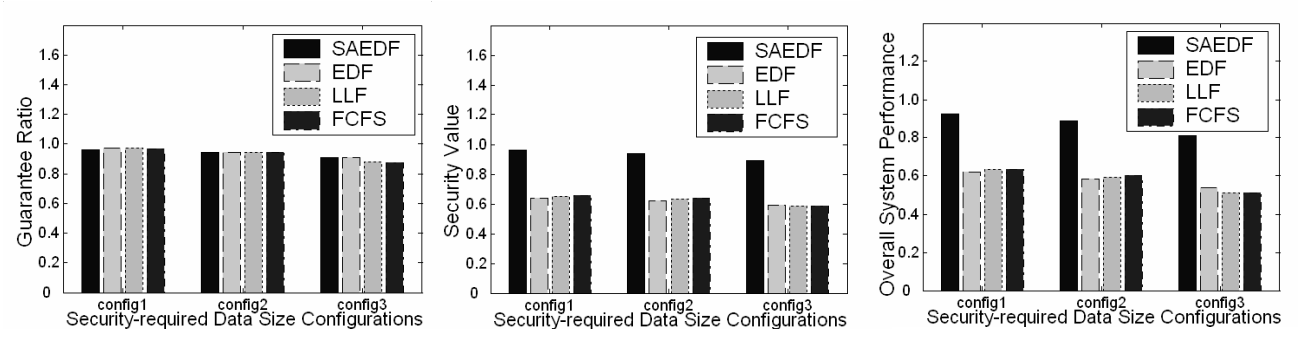


Fig. 17. 128 nodes control 64 F-16 aircrafts.

instances.

We simulated a 128-node system where each node has a 1000 MIPS CPU. We considered two node-to-aircraft configurations: (1) 128 nodes were used to control 128 aircrafts, and (2) 128 nodes controlled 64 aircrafts. In the first experimental setting, each node was responsible for controlling one aircraft; and in the second setting two nodes controlled one aircraft in a cooperative manner. The goal of the first case is to test the performance under a heavily loaded condition, whereas the second case is focused on the system performance under relatively light workloads.

The experimental results are shown in Fig. 16 and Fig. 17. Three observations are evident from Fig. 16. First, when the size of security-required data increases, guarantee ratio, security, and overall performance of the four algorithms noticeably decrease. This is because the security overhead rises with the increasing security-required data size. Consequently, the number of feasible tasks reduces, and the quality of security suffers.

Second, when 128 nodes are utilized to control 128 aircrafts, the guarantee ratio of SAEDF on average is only 79.84%, meaning that the system is slightly overloaded. In such a workload situation SAEDF consistently outperforms the other three algorithms in quality of security (see Fig. 16b) while maintaining the same guarantee ratio performance as EDF (see Fig. 16a). The results demonstratively show that SAEDF can maintain the same level of schedulability as EDF while significantly improving the security (by an average of 50.13%). More importantly, SAEDF significantly outperformed EDF, LLF and FCFS in overall system performance, which is the most crucial metric for security-critical real-time systems, by averages of 50.11%, 50.97% and 49.61%, respectively. The implication of this finding is that when system workloads are high, SAEDF can significantly improve overall system performance without adding extra hardware.

In the case where 128 nodes control 64 aircrafts, the average guarantee ratio of SAEDF is 93.93% (See Fig. 17). Under such a light workload condition, the guarantee ratios of SAEDF and the other three alternatives are almost identical. We attribute this result to the fact that when an aircraft is controlled by two computers, almost all the tasks dedicated to the aircraft can be accomplished before their deadlines because of the sufficient computational resources. In addition, the results presented in Fig. 17 indicate that although EDF and LLF can achieve 94.12% and 93.26% in guarantee ratio, their average security values are as low as 0.62. These results suggest that EDF and LLF are unsuitable for security sensitive applications. By using SAEDF as a security-aware scheduling heuristic, perfor-mance in security value is improved by an average of 50% over EDF and LLF.

7 Summary and Future Work

We presented in this paper a novel security-aware heuristic strategy (SAREC) for real-time applications on clusters. This strategy paves the way to the design of security-aware real-time scheduling algorithms. To make such security-aware scheduling algorithms practical, we proposed a security overhead model to quantitatively measure overheads of security services such as confidentiality, integrity, and authentication required by real-time applications. In doing so, security overheads can be taken into consideration in the process of scheduling real-time tasks. The effectiveness of the SAREC strategy was evaluated by implementing a novel security-aware real-time scheduling algorithm (SA-

EDF), which incorporates the earliest deadline first (EDF) scheduling algorithm into the SAREC strategy. SAREC is a general strategy in the sense that it can be applied to other existing real-time scheduling policies like LLF (see Section 6.8). To quantitatively validate the performance of the SA-EDF algorithm, we conducted extensive trace-driven simulations and introduced two new performance metrics, namely, *security value* (see Equation 4) and *overall system performance* (see Equation 16). Security value is a collective value of each accepted application's security level and it can be used to measure the quality of security experienced by all schedulable real-time tasks. Overall system performance, the most important performance metric for security-critical real-time systems, is a comprehensive metric defined as a product of security value and guarantee ratio. Experimental results based on real-world traces and a real application show that SAEDF achieves overall system performance over three existing scheduling algorithms (EDF, LLF, and FCFS) by average of 32.9%, 575.7%, and 713.6%, respectively. In addition, the empirical results reveal that SAEDF significantly improves quality of security for real-time tasks while maintaining high guarantee ratios under a wide range of workload characteristics.

Future studies in this research can be performed in the following directions.

1. Extend our security overhead models to multi-dimensional computing resources. For now, we simply consider CPU time, which is only one computing resource consumed by the security services. Memory, network bandwidth and storage capacities should be considered in the future.
2. Accommodate more security services into our security overhead model. Besides the three security services discussed, we plan to include authorization and auditing services into consideration.
3. Extend SAREC strategy to heterogeneous distributed systems. In a heterogeneous computing system, different nodes have different powers and resources. Thus, the same security requirement for a particular security service will result in different amount of overheads. A node-dependable security overhead calculating model should be developed.

ACKNOWLEDGMENT

This is a substantially revised and improved version of a preliminary paper [44] that appeared in the Proceedings of the 34th International Conference on Parallel Processing (ICPP'05), pages 5-12, June 2005. The revisions include a new security overhead model, new experimental results for performance of the SAEDF, EDF, LLF, and FCFS algorithms, six new sets of experiments, three properties and two theorems for the SAEDF algorithm, and performance evaluation with a real application. The work reported in this paper was supported in part by the New Mexico Institute of Mining and Technology under Grant 103295 and by Intel Corporation under Grant 2005-04-070. We are grateful to the anonymous referees for their insightful suggestions and comments.

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] 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,* 49(11), Nov. 2000, pp.1170-1183.

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

[4] A. Amin, R. Ammar, and A. El Dessouly, "Scheduling real time parallel structures on cluster computing with possible processor failures," *Proc. Int'l Symp. Computers and Comm.*, June 2004.

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

[6] 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.

[7] M. Bishop, Computer Security, ISBN 0-201-44099-7, *Addison-Wesley*, 2003.

[8] A. Bosselaers, R. Govaerts and J. Vandewalle, "Fast hashing on the Pentium," *Proc. Advances in Cryptology*, Springer-Verlag, 1996.

[9] 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*, Apr. 1999.

[10] S. Cheng and Y. Huang, "Dynamic real-time scheduling for multiprocessor tasks using genetic algorithm," *Proc. 28th Ann. Int'l Conf. Computer Software and Applications*, pp. 154 – 160, Sept. 2004.

[11] 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.

[12] 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, 2003.

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

[14] 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.

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

[16] 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.

[17] 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*, 2001.

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

[19] 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.

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

[21] V. Kalogeraki, P.M. Melliar-Smith, L.E. Moser, "Dynamic scheduling for soft real-time distributed object systems," *Proc. IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing*, pp.114-121, 2000.

[22] 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.

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

[24] A. K. Mok, "fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment," *Ph.D. Dissertation*, MIT, 1983.

[25] 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.

[26] E. Nahum, S. O'Malley, H. Orman and R. Schroeppel, "Towards High Performance Cryptographic Software," *Proc. IEEE Workshop Arch. and Implementation of High Perf. Comm. Subsys.*, August 1995.

[27] M. Pourzandi, I. Haddad, C. Levert, M. Zakrewski, and M. Dagenais, "A New Architecture for Secure Carrier-Class Clusters," *IEEE Int'l Workshop on Cluster Computing*, 2002.

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

[29] 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.

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

[31] 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.

[32] X. Qin and H. Jiang, "A Dynamic and Reliability-driven Scheduling Algorithm for Parallel Real-time Jobs on Heterogeneous Clusters," *Journal of Parallel and Distributed Computing*, Vol. 65, No. 8, pp.885-900, August 2005.

[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.*, 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] S. Song, Y.K. Kwok, and K. Hwang, "Trusted Job Scheduling in Open Computational Grids: Security-Driven Heuristics and A Fast Genetic Algorithms," *Proc. Int'l Symp. Parallel and Distributed Processing*, 2005.

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

[39] 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.

[40] 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.

[41] 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.

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

[43] T. Xie and X. Qin, "Enhancing Security of Real-Time Applications on Grids through Dynamic Scheduling," *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing* PP.146-158, MA, USA, June 19, 2005.

[44] T. Xie, X. Qin, and A. Sung, "SAREC: A Security-Aware Scheduling Strategy for Real-Time Applications on Clusters," *Proc. the 34th Int'l Conf. Parallel Processing*, Norway, June 14-17, 2005.

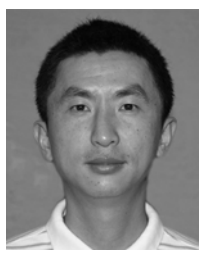
[45] T. Xie, X. Qin, A. Sung, M. Lin, and L. Yang, "Real-Time Scheduling with Quality of Security Constraints," *Int'l Journal of High Performance Computing and Networking*, February, 2006.

[46] W. Yurcik, X. Meng, G. A. Koenig, "A Cluster Process Monitoring Tool for Intrusion Detection: Proof-of-Concept," *Proc. 29th IEEE Conf. Local Computer Networks* 2004.

[47] W. Yurcik, X. Meng, G. Koenig, J.Greenseid "Cluster security as a unique problem with emergent properties", *The 5th LCI International Conference on Linux Clusters: The HPC Revolution 2004*, Austin, TX, May 18-20, 2004.

[48] X. Zhang, Y. Qu, and L. Xiao, "Improving Distributed Wrokload Performance by Sharing both CPU and Memory Resources," *Proc. 20th Int'l Conf. Distributed Computing Systems*, Apr. 2000.

[49] Y. Zhang, A. Sivasubramaniam, J. Moreira, and H. Franke, "Impact of workload and system parameters on next generation cluster scheduling mechanisms," *IEEE Trans. Parallel and Distributed Systems*, Vol. 12 , No. 9, pp. 967 – 985, Sept. 2001.



Tao Xie is a Ph.D. Candidate in Computer Science at the New Mexico Institute of Mining and Technology, USA. He received the BSc and MSc from Hefei University of Technology, China, in 1991 and 2000, respectively. His research interests are security-aware scheduling, high performance computing, cluster and Grid computing, parallel and distributed systems, real-time/embedded systems, and Information Security.



Xiao Qin received the BSc and MSc degrees in Computer Science from Huazhong University of Science and Technology, China in 1996 and 1999, respectively. He received his Ph.D. in Computer Science from University of Nebraska-Lincoln, USA, in 2004. He is an Assistant Professor in Department of Computer Science at the New Mexico Institute of Mining and Technology, USA. His research interests are parallel/distributed systems, real-time computing, storage systems, and performance evaluation. He has published more than 45 technical papers and served as program committee of several prestigious conferences, including The 35th International Conference on Parallel Processing, and The 25th IEEE International Performance Computing and Communications Conference.