Security-Aware Resource Allocation for Real-Time Parallel Jobs on Homogeneous and Heterogeneous Clusters

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

Abstract— Security is increasingly becoming an important issue in the design of real-time parallel applications, which are widely used in industry and academic organizations. However, existing resource allocation schemes for real-time parallel jobs on clusters generally do not factor in security requirements when making allocation and scheduling decisions. In this paper, we develop two resource allocation schemes, called TAPADS (Task Allocation for Parallel Applications with Deadline and Security constraints) and SHARP (Security- and Heterogeneity-Aware Resource allocation for Parallel jobs), by taking into account applications' timing and security requirements in addition to precedence constraints. We consider two types of computing platforms: homogeneous clusters and heterogeneous clusters. To facilitate the presentation of the new schemes, we build mathematical models to describe a system framework, security overhead, and parallel applications with deadline and security constraints. The proposed schemes are applied to heuristically find resource allocations that maximize the quality of security and the probability of meeting deadlines for parallel applications running on clusters. Extensive experiments using real world applications and traces as well as synthetic benchmarks demonstrate the effectiveness and practicality of the proposed schemes.

Index Terms— Security constraints, real-time scheduling, security overhead model, parallel jobs, clusters.

1 Introduction

OVER the past decade, clusters have become increasingly popular as powerful and cost-effective platforms for executing real-time parallel applications [27][28]. To improve their utilization and share their resources to outside users, more and more clusters are switching from traditional proprietary computing environments to open systems that are frequently exposed to public networks [23]. Consequently, they are subject to a variety of external attacks such as computing cycles stealing [23], inter-node communication snooping [17], and cluster services disruption [14]. Therefore, security mechanisms in the form of security services like authentication, integrity check, and confidentiality have been deployed on clusters to thwart the attacks [17][23]. These security services not only protect cluster computing platforms from being compromised by hackers [24], but also meet security requirements imposed by applications running on clusters [6][8].

Real-time parallel applications with security requirements running on clusters are emerging in many domains, including on-line transaction processing systems [2], medical electronics [9], aircraft control [1], and scientific parallel computing [6]. These applications propose various security requirements like data privacy [6], data integrity check [8], and software execution protection [24], and thus, are fundamentally distinguished by runtime uncertainties that are caused by security needs. For example, in parallel computing, protection of computationally expensive or irreplaceable data as well as valuable application software is critical [13]. Especially, in the business world and government, where the data is considered sensitive, the potential data losses due to a security incident could be catastrophic [13]. As a result, employing the security services provided by clusters is essential for security-critical real-time parallel applications.

Using security services to satisfy the applications' security needs, however, incurs security overhead in terms of computation time, which might violate the applications' deadlines. The conflicting requirements of good real-time performance and high quality of security protection imposed by security-critical real-time applications introduce a new challenge for resource allocation schemes, i.e., how to solve the real-time and security dilemma. Moreover, security heterogeneity (see Section 5.1) existed in heterogeneous clusters makes solving this dilemma more difficult as security overhead is node-dependent, which means for the same level of security service different computing nodes incur distinct security overhead. Unfortunately, existing resource allocation schemes for real-time parallel applications on clusters [27][28] normally do not factor in applications' security requirements when making resource allocation decisions, and thus, are inadequate for security-critical real-time parallel applications. Hence, security-aware resource allocation schemes must be developed to bridge the gap between the incapability of existing schemes and the needs

- *T. Xie is with the Department of Computer Science, San Diego State University, San Diego, CA 92182. E-mail: xie@cs.sdsu.edu.*
- *X. Qin is with the Department of Computer Science and Software Engineerin, Auburn University, Auburn, AL 36849. E-mail: xqin@auburn.edu.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

Digital Object Indentifier 10.1109/TPDS.2007.70776 1045-9219/$25.00 © 2007 IEEE

of high quality of security demanded by security-critical real-time applications. Motivated by this discrepancy, in this paper, we design and evaluate two security-aware resource allocation schemes called TAPADS (Task Allocation for Parallel Applications with Deadline and Security constraints) and SHARP (Security- and Heterogeneity-Aware Resource allocation for Parallel jobs) for real-time parallel applications running on homogeneous and heterogeneous clusters, respectively. While TAPADS is developed for parallel applications represented by DAGs (Directed Acyclic Graph) where precedence constraints and communications among tasks in an application exist, SHARP is dedicated for embarrassingly parallel applications with no such precedence constraints and communications [37]. To the best of our knowledge, TAPADS and SHARP are the first two security-aware resource allocation strategies for real-time parallel applications running on clusters. The fundamental contributions of this paper include the following three aspects:

- Design and evaluation of two resource allocation schemes for real-time parallel jobs with security constraints running on homogeneous and heterogeneous clusters. Extensive experiments using synthetic workloads, traces, and real-world applications validate the effectiveness of the two security-aware resource allocation strategies.
- Proposition of a security overhead model that can be used to quantitatively measure security overhead in terms of computation time introduced by security services.
- Investigation of the impacts of heterogeneities on real-time performance and quality of security.

The rest of the paper is organized as follows. We summarize related work and our new approach in the next section. Section 3 describes the system architecture and security overhead model. In Section 4, we propose the security-aware allocation scheme for homogeneous clusters. Section 5 presents the resource allocation scheme for security sensitive and real-time applications on heterogeneous clusters. Section 6 concludes the paper with summary and future directions.

2 Related Work and Our New Approach

In this section, first we discuss related work on parallel job scheduling, cluster security techniques, and trade-offs between real-time performance and security. Next, we introduce our new approach to solving the security and real-time dilemma for security-critical real-time parallel applications.

2.1 Related Work

Since allocation and scheduling parallel jobs onto a set of processors generally fall into the class of NP-complete problems [11], the scheduling problem modelled in this paper is NP-complete as well because it is essentially a general problem of scheduling parallel jobs onto a set of processors plus one more constraint in satisfying security requirements of parallel jobs. Thus, heuristic scheduling algorithms become practical solutions to the problem.

The issue of allocating and scheduling real-time applications using heuristic approaches has been thoroughly studied [1][19][26]. Normally, the goal of these heuristic algorithms is to improve real-time performance by decreasing the number of jobs whose deadlines are missed. Hou and Shin proposed a resource allocation scheme for periodic tasks with precedence constraints in distributed real-time systems [12]. He et al. studied the problem of dynamic scheduling of parallel real-time jobs executing on heterogeneous clusters [10]. These schemes provide high schedulability for real-time systems. However, they are not suitable for security-sensitive real-time parallel applications due to their oversight and ignorance of security requirements imposed by the applications.

Security concerns on clusters attracted attentions from researchers in recent years. A vast variety of security techniques have been developed for clusters [3][6][17][23]. Connelly and Chien addressed the issue of protecting tightly coupled, high-performance component communication [6]. Apvrille and Pourzandi proposed a new security policy language named distributed security policy, or DSP, for clusters [3]. Although the above security techniques are not developed for solving the issue of scheduling real-time applications, the security services that they provided can be exploited by security-critical real-time parallel applications to satisfy their security needs.

Since the utilization of security services causes extra overhead in terms of computation time, a security overhead model that quantitatively measures security overhead for commonly used security services is essential for a security-aware resource allocation scheme. Unfortunately, the only previous work on measuring security cost was a preliminary method for defining the costs associated with network security services proposed by Irvine and Levin [16]. Even so, they only illustrated three simple security cost examples without offering a feasible security overhead measurement model.

The closest work to this research reported in the literature was accomplished by Song et al. very recently [25]. They developed three risk-resilient strategies and a genetic algorithm based scheme STGA (Space-Time Genetic Algorithm) to provide security assurance in Grid job scheduling. However, their algorithms cannot be applied on clusters for real-time parallel applications with security requirements. First, their algorithms are unable to support real-time applications as Grid jobs can hardly have real-time constraints. Next, their algorithms only consider batch scheduling where jobs are independent from each other, and thus, cannot schedule parallel jobs where precedence constraints and communications among tasks within one job exist.

2.2 Our New Approach

Our work is built upon the related work on cluster security, security overhead measurement, and real-time parallel job scheduling. Since snooping, alteration, and spoofing are three common attacks in cluster environments [23], we considered three security services, authentication service, integrity service, and confidentiality service, to

guard clusters. For example, snooping, an unauthorized interception of information, can be countered by confidentiality services, which encrypt data by using cryptographic algorithms so that a hacker cannot correctly interpret the data [4]. We assume that the three security services are available to security-critical real-time parallel jobs submitted to a cluster where a security-aware resource allocation scheme is applied. How the three security services are provided to parallel jobs can be found in our previous work [35][36]. Security services usually consume multiple computing resources like computation time, memory, bandwidth, and storage capacities. However, in real-time job scheduling, computation time is the most important security overhead because it delays jobs' completion times, which in turn could violate their deadlines. Hence, we only consider security overhead in terms of computation time in this work, and leave the investigation of the impacts of rest types of security overhead on security-aware resource management in our future work. Thereafter, security overhead means computation time caused by security services.

Two formats of security requirement specification, a single security level for each required security service and a security range for each required security service, are used in this work. A security level is the strength or safety degree of a particular security service. Normally, a security level of a security service corresponds to a particular security mechanism because different security mechanisms provide distinct security strengths. Basically, a security level is a normalized value when setting the strongest security mechanism as 1. A security range is a scope that contains multiple distinct security levels for a particular security service. The lowest value in a security range indicates the minimal security strength mandated by the user, while the highest value implies the maximal security strength necessary for the user and all the values above should not be considered. The single security level format is suitable for situations where each job only demands a baseline (minimal) security level for each security service required, whereas the security range format is adequate for scenarios where a security level higher than the maximal value in the security range is not necessary for a job due to the job's relatively low importance or the user's tight budget.

Parallel applications generally fall into two camps, non-embarrassingly parallel applications represented by DAGs and embarrassingly parallel applications. We considered both in this work with TAPADS for DAGs and SHARP for embarrassingly parallel applications. Each parallel application consists of multiple tasks that share a common deadline, which is the deadline of the entire application. Each task in an application demands an array of security services with different levels. For security requirements in a security range format, TAPADS verifies whether the application's deadline can be met with all its tasks' minimal security levels for all required security services satisfied. If so, TAPADS further optimizes the tasks' security levels within the security range under the condition that the security level enhancements will not result in the application's deadline to be violated. Otherwise, the job will be dropped because its execution is unsafe. For each task, SHARP discovers all computing nodes that can meet its deadline. If no such node can be found for a task, the entire application will be aborted. If each task has one or multiple nodes that can meet the application's deadline, SHARP assigns the task onto a node that can minimize the degree of security deficiency.

3 PRELIMINARIES

We describe in this section mathematical models, which were built to represent a resource allocation framework and security overhead. For future reference, we summarize notations used in this study in Table 1.

3.1 Resource Allocation Architecture

As depicted in Fig. 1, a cluster is comprised of m nodes connected via a high-performance network to process parallel applications submitted by users. Note that throughout this paper the terms application and job are used interchangeably. Let $Y = \{y_1, y_2, \ldots, y_m\}$ denote the set of m nodes in a cluster. Each node communicates with other nodes through message passing, and the communication time between two tasks assigned to the same node is assumed to be negligible. Note that the communication subsystem, an underlying communication infrastructure of a cluster, supports messages with time constraints, meaning that the worst-case link delay is predictable and bounded. Examples of such real-time communication subsystems can be found in the literature [38]. Additionally, the communication subsystem considered in our study provides full connectivity in a way that any two nodes are connected through either a physical link or a virtual link. This assumption is arguably reasonable for modern interconnection networks (e.g. Myrinet [29]) that are widely used in high-performance clusters.

The resource allocation architecture consists of a security-aware task allocator, an admission controller, and a real-time scheduler. The security-aware task allocator is intended to generate resource allocation decision for each task of a parallel application, satisfying both security and real-time requirements. The admission controller is deployed to perform feasibility checks by determining if arriving parallel applications can be completed by a cluster before their specified deadlines. An application will be admitted into the system if its deadline can be met. The scheduler is to satisfy timing requirements of parallel applications by assigning high priorities to jobs with early deadlines.

3.2 Security Overhead Model

For each security service, we assume that there are several alternative security methods or algorithms, which can be used to accomplish the service. More precisely, we assume that three authentication methods HMAC-MD5, HMAC-SHA-1, and CBC-MAC-AES are available for users to select to fulfil the authentication service. Similarly, we assume that seven hash functions (MD4, MD5, RIPEMD, RIPEMD-128, SHA-1, RIPEMD-160 and Tiger) and eight encryption algorithms (SEAL, RC4, Blowfish, Knufu/Khafre, RC5, Rijndael, DES and IDEA) are pro

TABLE 1. DEFINITIONS OF NOTATIONS

Notation	Definition	Notation	Definition
T	A set of n non-preemptable real-time tasks	E	A set of weighted and directed edges representing communications among tasks in T
p	Period, a constant time interval between two successive job instances of a parallel application J	J	A non-embarrassingly parallel application, $J = (T, E, p)$
t_i	The ith task in the task set T ($1 \le i < n$)	e_i	The execution time of t_i without security overhead
l_i	The amount of security-sensitive data (KB) of task t_i	S_i	The security requirement vector of task t_i
S_i^j	An element of S_i, the security level range of the jth security service required by t_i	s_i^j	The security level of the jth ($j \in \{a, e, g\}$) security service required by task t_i
$c_i^j(s_i^j)$	The security overhead of the jth ($j \in \{a, e, g\}$) security service imposed by task t_i	$\mu^g(s_i^g)$	A function mapping an integrity security level to its corresponding hash function's performance
$\sigma^e(s_i^e)$	A function mapping a confidentiality security level to its corresponding encryption algorithm's performance	$\eta^a(s_i^a)$	A function mapping an authentication security level to its corresponding authentication method's performance
a, e, g	Authentication, Confidentiality, Integrity	m_k	The kth computing node in a cluster
e_{ij}	The volume of data transmitted between task t_i (on node m_k) and task t_j (on node m_a)	b_{ka}	The bandwidth between node m_k and node m_a
$SL(s_i)$	The security benefit of task t_i (see 5)	$SV(T,E)$	*Security Value*, the total security benefit gained by a parallel application J (see 12)
$P_{SD}(X)$	The probability that all tasks meet deadline constraints under an allocation scheme X (see 23)	$P_{SC}(X)$	*QSA*, the quality of security of application J under an allocation scheme X (see 31)
$P_C(X)$	The probability that all tasks are free from being attacked under an allocation scheme X (see 26)	$P_L(X)$	The probability that messages are risk-free under an allocation scheme X (see 30)
H^C	The computational heterogeneity of an embarrassingly parallel application (see 33)	H^V	The heterogeneity in security services of the cluster (see 38)
$DSD(T,X)$	The sum of security deficiency values of all tasks in an embarrassingly parallel application (see. 44)	P_{rf}	Risk-free probability of a heterogeneous cluster (see 53)
w_i^j	The weight of the jth security service for task t_i (see 5)	λ_i^j	Risk rate caused by the jth security service required by task t_i (see 24)

vided for users to realize the integrity service and the con fidentiality service, respectively. In what follows, we first give a general expression of our security overhead model. Next, we use the integrity service as an example to show how we assign security levels to different security mechanisms and how we calculate security overhead for each security service. Last, we justify the feasibility of our security overhead m
odel.

We assume that task t_i requires all of the three security services provided in a sequential order. Let s_i^j and $c_i^j(s_i^j)$ be the security level and the overhead of the jth ($j \in$ {a, e, g}) security service, the security overhead c_i experienced by t_i, can be computed using (1).

$$c_i = \sum_{j \in \{a, e, g\}} c_i^j(s_i^j), s_i^j \in S_i^j, \quad (1)$$

$c_i^e(s_i^e)$, $c_i^g(s_i^g)$, and $c_i^a(s_i^a)$ are overheads caused by the confidentiality, integrity, and authentication services [33]. S_i^j denotes task t_i's required security level range of the jth service.

The performance of the seven hash functions is listed in Table 2. For example, 23.90 KB/ms means for every millisecond the hash function MD4 can process 23.90 KB data. Based on its 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, and security levels for the other hash functions can be calculated by (2), where μ_i^g is the performance of the ith ($1 \le i \le 7$) hash function.

$$s_i^g = 4.36/\mu_i^g, 1 \le i \le 7. \quad (2)$$

For example, the security level of hash function RIPEMD is 0.36 because 4.36/12 is about 0.36. Let s_i^g be the integrity security level of task t_i, and the overhead of the integrity service can be calculated using (3), where l_i is the amount of data whose integrity must be achieved, and $\mu^g(s_i^g)$ is a function used to map a security level to its corresponding hash function's performance.

$$c_i^g(s_i^g) = l_i/\mu^g(s_i^g). \quad (3)$$

For instance, the output of the function $\mu^g(s_i^g)$ is 4.36 KB/ms when the input s_i^g is 1. Similarly, let s_i^e be the confidentiality security level of task t_i, and the computation overhead

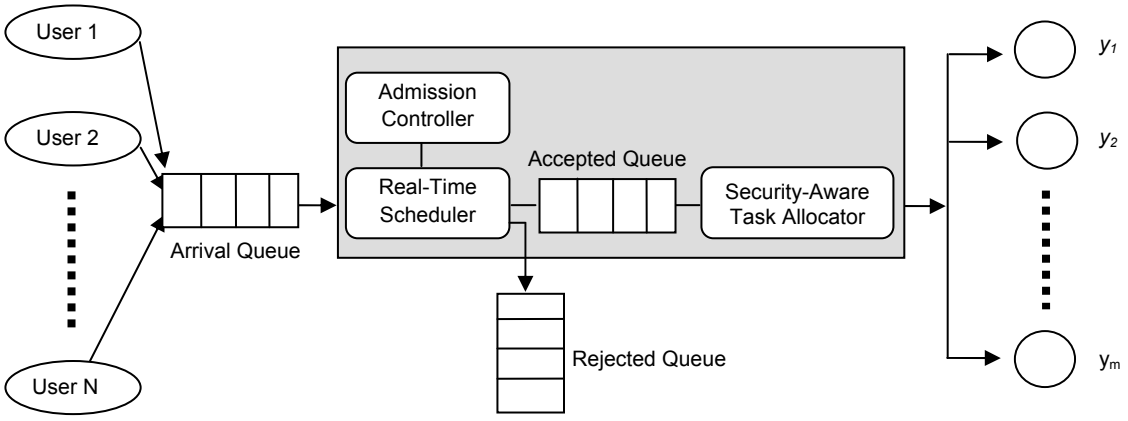


Fig. 1. Security-aware resource allocation architecture.

of a selected confidentiality service can be calculated using (4), where l_i is the amount of data whose confidentiality must be guaranteed, and $\sigma^e(s_i^e)$ is a function used to map a security level to its corresponding encryption method's performance. The function $\sigma^e(s_i^e)$ was defined in Table 1 Cryptographic Algorithms for Confidentiality in our previous work [33], where the performance and the security levels of the eight encryption algorithms were summarized.

$$c_i^e(s_i^e) = l_i / \sigma^e(s_i^e). \quad (4)$$

Since the security overhead of a particular authentication method is a constant value, the security overhead of authentication service $c_i^a(s_i^a)$ is equal to its performance $\eta^a(s_i^a)$. The function $\eta^a(s_i^a)$ can be derived from the Table 3 in [33].

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

In fact, how to quantitatively measure security is still an open question [5][16][18][22] and is out of the scope of this paper. The security model used in this paper is only a step towards finding a way to quantitatively approximate relative strength of some commonly used security mechanisms. We believe that our model is reasonable in this research due to the following three reasons. First, the fundamental assumption of our security overhead model is valid. Our assumption is that people only accept a slower security mechanism if and only if it can provide a higher level security compared with its faster peers. Although the strength of some cryptographic schemes could be orthogonal to their processing overhead, this assumption is generally safe because many security mechanisms can achieve a higher amount of security by doing more computations [5]. For example, the strength of encryption schemes depends on the size of the key and the number of encryption rounds [21]. Larger key sizes or number of rounds result in higher levels of security at the cost of additional computation time [5]. Therefore, the way we assign different security mechanisms with distinct security levels based on their performance is reasonable. Second, although the measurements of security requirements and security levels are not completely objective, the improvements of our TAPADS and SHARP algorithms compared with the existing approaches in terms of security are still valid because all algorithms were evaluated by using the same set of security calculation criteria under the same circumstance. Third, quantitatively modeling security requirements and security levels makes it possible for us to compare the security performance of different algorithms and perceive the differences among them.

4 SECURITY-AWARE RESOURCE ALLOCATION SCHEME FOR HOMOGENEOUS CLUSTERS

4.1 Task Model

4.1.1 Deadline and Precedence Constraints

Applications with dependent real-time tasks can be modelled by Directed Acyclic Graphs (DAGs) [15]. Throughout this paper, a parallel application is defined as a vector $J = (T, E, p)$, where $T = \{t_1, t_2, ..., t_n\}$ represents a set of non-preemptable real-time tasks, E is a set of weighted and directed edges used to represent communication among tasks, e.g., $(t_i, t_j) \in E$ is a message transmitted from task t_i to t_j, and p is the period, a constant time interval between two successive job instances of the parallel application J. Precedence constraints of the parallel application is represented by all the edges in E. Communication time for sending a message $(t_i, t_j) \in E$ from task t_i on node m_k to task t_j on node m_a is determined by e_{ij}/b_{ka}, where e_{ij} is the volume of data and b_{ka} is the bandwidth between m_k and m_a. A task is characterized by three parameters, e.g., $t_i = (e_i, l_i, S_i)$, where e_i is the execution time, l_i denotes the amount of data (measured in KB) to be protected, and S_i is a vector of security requirements (see section 4.1.2.). Note that e_i can be estimated by code profiling techniques and it does not include security overhead.

This study is focused on the issue of allocating periodic jobs on clusters. A parallel application generates a sequence of job instances $J_i^0, J_i^1, J_i^2 ...$, where J_i^j must be finished before J_i^{j+1} can start executing. Note that there is a constant interval between two consecutive job instances. The deadline of J_i^j is the arrival time of the next task instance. Although the arrival time of a task instance is not explicitly specified in the model, the arrival time can be determined when the task instance is released dynamically during the system execution. It has been proved that there exists a feasible schedule for a set of periodic tasks if and only if there is a feasible schedule for the *planning cycle* of the tasks [12]. Note that the planning cycle is the least common multiple of all the tasks' periods. Thus, the behaviour of the set of periodic tasks can be effectively analysed within the planning cycle.

4.1.2 Security Constraints

A collection of security services required by task t_i is specified as $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, where S_i^j represents the required security level range of the jth security service. Our allocation schemes aim at determining 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 real-world example of real-time applications with various levels of security requirements was illustrated in [32].

In an effort to maximize quality of security, the resource allocation schemes have to measure security benefits gained by a parallel application. We model the security benefit of the ith task in T as a security level function denoted by $SL: S_i \to \Re$, where $\Re$ is the summation of a set of positive real numbers:

$$SL(s_i) = \sum_{j=1}^{q} w_i^j s_i^j, \quad (5)$$

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

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

$$SL(T) = \sum_{i=1}^{n} SL(s_i), \text{ wehre } s_i = \left(s_i^1, s_i^2, ..., s_i^q\right) \quad (6)$$

We can obtain the following non-linear optimization problem formulation for a task set *T*:

$$\text{maximize } SL(T) = \sum_{i=1}^{n}\sum_{j=1}^{q} w_i^j s_i^j,$$
$$\text{subject to } \min(S_i^j) \le s_i^j \le \max(S_i^j), \quad (7)$$

where $\min(S_i^j)$ and $\max(S_i^j)$ are the minimum and maximum security requirements of task t_i.

An array of security services required by message $(t_i, t_j) \in E$ is specified as $\hat{S}_{ij} = (\hat{S}_{ij}^1, \hat{S}_{ij}^2, ..., \hat{S}_{ij}^p)$, where $\hat{S}_{ij}^k$ denotes the required security level range of the *k*th security service. The most appropriate point $\hat{s}_{ij}$ in space $\hat{S}_{ij}$ has to be calculated, e.g., $\hat{s}_{ij} = (\hat{s}_{ij}^1, \hat{s}_{ij}^2, ..., \hat{s}_{ij}^p)$, where $\hat{s}_{ij}^k \in \hat{S}_{ij}^k$, $1 \le j \le q$ We model the security benefit of the message as the following function:

$$SL(\hat{s}_{ij}) = \sum_{k=1}^{p} \hat{w}_{ij}^k \hat{s}_{ij}^k \quad (8)$$

where $\hat{s}_{ij} = (\hat{s}_{ij}^1, \hat{s}_{ij}^2, ..., \hat{s}_{ij}^p), 0 \le \hat{w}_{ij}^k \le 1$, and $\sum_{j=1}^{p} \hat{w}_{ij}^k = 1$.

Note that weight $\hat{w}_{ij}^k$ reflects relative priorities of the *k*th required security service. The security benefit of a message set is calculated as the summation of the security levels of all the messages.

$$SL(E) = \sum_{(t_i, t_j)\in E} SL(\hat{s}_{ij}), \text{ where } \hat{s}_{ij} = (\hat{s}_{ij}^1, \hat{s}_{ij}^2, ..., \hat{s}_{ij}^p). \quad (9)$$

The optimal security benefit of the message set can be computed as follows:

$$\text{maximize } SL(E) = \sum_{(t_i, t_j)\in E}\sum_{k=1}^{p} \hat{w}_{ij}^k \hat{s}_{ij}^k,$$
$$\text{subject to } \min(\hat{S}_{ij}^k) \le \hat{s}_{ij}^k \le \max(\hat{S}_{ij}^k), \quad (10)$$

where $\min(\hat{S}_{ij}^k)$ and $\max(\hat{S}_{ij}^k)$ are the minimum and maximum security requirements of the message.

Now we can define an optimization problem formulation to compute an optimal security benefit of a parallel application, subject to certain timing and security constraints:

$$\text{maximize } SV(T,E)=SL(T)+SL(E). \quad (11)$$

Substituting (7) and (10) into (11) yields the following security value objective function

$$SV(T,E) = \sum_{i=1}^{n}\sum_{k=1}^{q} w_i^k s_i^k + \sum_{(t_i, t_j)\in E}\sum_{k=1}^{p} \hat{w}_{ij}^k \hat{s}_{ij}^k, \quad (12)$$

4.2 The TAPADS Algorithm

This section presents a resource allocation algorithm (TAPADS) for homogeneous clusters. Let *X* be an *m* by *n* binary matrix corresponding to an allocation, in which *n* tasks are assigned to *m* nodes in the cluster. Element x_{ij} equals to 1 if and only if t_i has been allocated to node m_j; otherwise $x_{ij} = 0$.

The algorithm outlined in Fig. 2 aims at achieving high security under two conditions: (1) increasing security levels will not result in missing deadlines; and (2) precedence constraints are satisfied. To meet deadline and precedence constraints, TAPADS assigns tasks to nodes in a way to maximize security measured by $P_{SC}(X)$, which is the probability that all tasks are executed without any risk of being attacked and all messages are risk-free during the course of transmissions. Further, TAPADS can

1. Sort and renumber tasks so that if $(t_i, t_j) \in E$ then $i < j$;
2. Compute the critical path of the task graph;
$$f = \sum_{t_i \in \text{critical path}} (e_i + c_i^{\min})$$
3. Calculate the tentative finish time, f, where ei is the computation time of task ti, and c_i^{min} is the security overhead of task t_i when the minimal security requirements of t_i are met.
4. Allocate and schedule all t_i in the critical path subject to minimal security requirements;
5. Allocate and schedule all $t_i \notin$ critical path subject to precedence and minimal security constraints;
6. Obtain slack time, *slk* = *d* – *f*, where *d* is the deadline;

while (slack time ≥ 0) **do**

7.1 select *i'* and *j'* subject to
$$w_{i'}^{j'} \Delta s_{i'}^{j'} \Big/ \left(c_{i'}^{j'}(s_{i'}^{j'} + \Delta s_{i'}^{j'}) - c_{i'}^{j'}(s_{i'}^{j'})\right) = \max_{1\le i\le n, 1\le j\le q} \left\{ w_i^j \Delta s_i^j \Big/ \left(c_i^j(s_i^j + \Delta s_i^j) - c_i^j(s_i^j)\right)\right\}$$

7.2 select *a'* and *b'* subject to
$$\hat{w}_{a'b'}^{k'} \Delta \hat{s}_{a'b'}^{k'} \Big/ \left(\hat{c}_{a'b'}^{k'}(\hat{s}_{a'b'}^{k'} + \Delta \hat{s}_{a'b'}^{k'}) - \hat{c}_{a'b'}^{k'}(\hat{s}_{a'b'}^{k'})\right) = \max_{\substack{(t_a, t_b)\in E \\ 1\le k\le p}} \left\{ \hat{w}_{ab}^k \Delta \hat{s}_{ab}^k \Big/ \left(\hat{c}_{ab}^k(\hat{s}_{ab}^k + \Delta \hat{s}_{ab}^k) - \hat{c}_{ab}^k(\hat{s}_{ab}^k)\right)\right\}$$

7.3 **if** $w_{i'}^{j'} \Delta s_{i'}^{j'} \Big/ \left(c_{i'}^{j'}(s_{i'}^{j'} + \Delta s_{i'}^{j'}) - c_{i'}^{j'}(s_{i'}^{j'})\right) > \hat{w}_{a'b'}^{k'} \Delta \hat{s}_{a'b'}^{k'} \Big/ \left(\hat{c}_{a'b'}^{k'}(\hat{s}_{a'b'}^{k'} + \Delta \hat{s}_{a'b'}^{k'}) - \hat{c}_{a'b'}^{k'}(\hat{s}_{a'b'}^{k'})\right)$ **then**
increase security level $s_{i'}^{j'} \leftarrow s_{i'}^{j'} + \Delta s_{i'}^{j'}$ if $s_{i'}^{j'} < \max\{S_{i'}^{j'_l}\}$;

7.4 **else** increase security level $\hat{s}_{a'b'}^{k'} \leftarrow \hat{s}_{a'b'}^{k'} + \Delta \hat{s}_{a'b'}^{k'}$ if $\hat{s}_{a'b'}^{k'} < \max\{\hat{S}_{a'b'}^{k'_l}\}$;

7.5 update task allocation, e.g., start times of tasks and messages;

7.6 update slack time based on the increased security level;

end while

Fig. 2. The TAPADS algorithm.

maintain a high schedulability measured by $P_{SD}(X)$, which is the probability that all tasks are timely completed. Note that $P_{SC}(X)$ and $P_{SD}(X)$ will be derived in Section 4.3.

Before optimizing the security level of each task and message of a job, TAPADS makes the best effort to satisfy the deadline and precedence constraints. This can be accomplished by calculating the earliest start time and the minimal security overhead of each task and message in Steps 4 and 5. If the deadline can be guaranteed provided that the minimal security requirements are met, the slack time of the initial allocation can be obtained by Step 6.

To efficiently improve the quality of security of the job, in Step 7 TAPADS chooses the most appropriate task or message in which the security level will be increased. Specifically, it is desirable to give higher priorities to security services with higher weights and lower security overhead. Hence, we define the following two benefit-cost ratio functions, e.g., θ_i^j and $\hat{\theta}_{ij}^k$, which measure the increase of security level by unit security overhead.

$$\theta_i^j = w_i^j \Delta s_i^j \Big/ \left(c_i^j(s_i^j + \Delta s_i^j) - c_i^j(s_i^j)\right), \text{ for the } j\text{th service of } \mathrm{t}_i, \quad (13)$$

$$\hat{\theta}_{ij}^k = \hat{w}_{ij}^k \Delta \hat{s}_{ij}^k \Big/ \left(\hat{c}_{ij}^k(\hat{s}_{ij}^k + \Delta \hat{s}_{ij}^k) - \hat{c}_{ij}^k(\hat{s}_{ij}^k)\right), \text{ for the } k\text{th service of } (t_i, t_j), \quad (14)$$

where the numerators represent the weighted increase in the security level, whereas the denominators indicate the corresponding increase in security overhead.

After performing Steps 7.1 and 7.2, TAPADS identifies the best candidate in $T \cup E$ that has the highest benefit-cost ratio. Formally, the best candidate is chosen based on the following expression,

$$\begin{cases} \theta_{i'}^{j'} = \max\limits_{1\le i\le n, 1\le j\le q}\left\{\theta_i^j\right\}, \text{if } \max\limits_{1\le i\le n, 1\le j\le q}\left\{\theta_i^j\right\} \ge \max\limits_{(t_i,t_j)\in E, 1\le k\le p}\left\{\hat{\theta}_{ij}^k\right\}, \\ \theta_{i'j'}^{k'} = \max\left\{\max\limits_{(t_i,t_j)\in E, 1\le k\le p}(\hat{\theta}_{ij}^k)\right\}, \quad \text{otherwise.} \end{cases} \quad (15)$$

To yield a maximized security level of the job, Steps 7.3 and 7.4 are responsible for increasing security levels of more important services at the minimal cost. Thus, the slack time is distributed on a task or message with the highest benefit-cost ratio. Step 7.5 updates the task allocation in accordance with the increased security level, because start times of other tasks and messages are dependent of how the slack time is distributed. Finally, step 7.6 updates the slack time.

Theorem 1. *The time complexity of TAPADS is $O(k(q|T|+p|E|))$, where k is the number of times Step 7 is repeated, q is the number of security services for computation, p is the number of security service for communication. $|T|$ is the number of nodes (tasks) in a DAG and $|E|$ is the number of directed edges in a DAG.*

Proof. The time complexity of allocating and scheduling tasks subject to precedence and minimal security constraints is $O(|T|+|E|)$ (Steps 1-6). To effectively boost security levels of tasks and messages under the constraints (Steps 7.3-7.4), it takes time $O(|T|+|E|)$ to select the most appropriate task or message as a candidate whose quality of security will be improved. The time complexity of step 7 becomes $O(k(q|T|+p|E|))$. Thus, the time complexity of TAPADS is: $O(|T|+|E|)$ + $O(k(q|T|+p|E|)) = O(k(q|T|+p|E|))$. □

k cannot be very big number in practice, because k in many cases is much smaller than $|T|+|E|$. Therefore, the time complexity of TAPADS is reasonably low based on the expression above

4.3 Evaluation of Timeliness and Security Risks

In this section, we first explain a way in which tasks are allocated to nodes subject to precedence constraints. Then, we derive the probability $P_{SD}(X)$ that all tasks meet their deadline constraints. Finally, we calculate the probability $P_{SC}(X)$ that all tasks and messages are risk-free during the execution of the job. It is to be noted that $P_{SD}(X)$ and $P_{SC}(X)$ help in evaluating the performance of our algorithm in Section 4.4.

4.3.1 Task and Message Scheduling

The proposed allocation scheme relies on the way of scheduling tasks and messages, which in turn depend on the values of two important parameters: (1) *est(t)*, the earliest start time for task *t*, and (2) *eat(t)*, the earliest available time for *t*. Although both *est(t)* and *eat(t)* indicate a time when task *t*'s precedence constraints have been met (i.e. all messages from *t*'s predecessors have arrived), *est(t)* additionally signifies that node *m(t)* (to which *t* is allocated) is now available for *t* to start execution. Thus, *est (t)* ≥ *eat(t)*, and at time *eat(t)* node *m(t)* may not be ready for *t* to execute. In what follows, we derive the expressions of *eat(t)* and *est(t)* needed for scheduling tasks and messages.

If task t_i had only one predecessor task t_j, then the earliest available time $eat_k(t_j, t_i)$ on the kth node is given by the following expression, where $f(t_j)$ is the finish time of t_j, $mst_{uv}(t_j, t_i)$ is the earliest start time of message (t_j, t_i), d_{ji} is the data volume, B_{uv} is the network bandwidth, and d_{ji}/B_{uv} is the transmission time for the message. Note that t_j and t_i are allocated to the uth and vth nodes.

$$eat_k(t_j, t_i) = \begin{cases} f(t_j) & \text{if } m(t_i) = m(t_j) \\ mst_{uv}(t_j, t_i) + d_{ji}/B_{uv} & \text{otherwise} \end{cases} \quad (16)$$

$mst_{uv}(t_j, t_i)$ depends on how the message is scheduled on the links. A message is allocated to a link if the link has an idle time slot that is later than the sender's finish time and is large enough to accommodate the message. Task t_i must wait until the last message from all its predecessors has arrived. Hence, the earliest available time of t_i is the maximum of $eat_k(t_j, t_i)$ over all its predecessors.

$$eat_k(t_i) = \max_{(t_j,t_i)\in E}\{eat_k(t_j, t_i)\}. \quad (17)$$

With (17) in place, we can obtain the earliest start time $est_j(t_i)$ on the jth node by checking if the node has an idle time slot that starts later than task's $eat_j(t_i)$ and is large enough to accommodate the task. $est_j(t_i)$ is a parameter used to derive $est(t_i)$, the earliest start time for the task on any node. Expression for $est(t_i)$ is given below.

$$est(t_i) = \min_{M_j\in M}\{est_j(t_i)\}. \quad (18)$$

4.3.2 Calculation of $P_{SD}(X)$

We now calculate the probability that all tasks meet deadline constraints under allocation X. It is worth noting that the initial allocation of X must satisfy the following timing constraint property:

$$\forall 1 \le i \le n : est(t_i) + e_i + c_i^{min} \le d, \quad (19)$$

where $est(t_i)$ is the time obtained from expression (18), d is the deadline, e_i is the computation time, and c_i^{min} is the security overhead when the minimal security requirements are met. c_i^{min} can be calculated by the following equation:

$$c_i^{min} = \sum_{j=1}^{q} c_i^j \left(\min\left\{S_i^j\right\}\right). \quad (20)$$

Based on an initial task allocation, the scheme judiciously raises security levels of tasks and messages provided that the following property is satisfied.

$$\forall 1 \le i \le n : est(t_i) + e_i + \sum_{j=1}^{q} c_i^j(s_i^j) \le d. \quad (21)$$

The allocation X is feasible if all tasks can be completed before the deadline. Therefore, the probability $P_{SD}(\text{tasks are timely completed under } X \text{ with } \bar{s})$ is expressed as below:

$$\varphi_i(s_i) = 1 \text{ , for } est(t_i) + e_i + \sum_{j=1}^{q} c_i^j(s_i^j) \le d$$

$$\varphi_i(s_i) = 0 \text{ , otherwise, where } \bar{s} = (s_1, s_2, ..., s_n),$$

$$P(\text{tasks are timely completed under } X \text{ with } \bar{s}) = \prod_{i=1}^{n} \varphi_i(s_i), \quad (22)$$

Under allocation X, the probability that all tasks are timely completed can be computed as:

$$P_{SD}(X) = \sum_{all\ \bar{s}_k} p_k P(\text{tasks are timely completed under } X \text{ with } \bar{s}_k)$$

$$= \sum_{all\ \bar{s}_k} p_k \prod_{i=1}^{n} \varphi_i(s_{ki}). \quad (23)$$

where the security level vector is represented as $\bar{s}_k = (s_{k1}, s_{k2}, ..., s_{kn})$, and p_k is the probability that the security level vector is $\bar{s}_k$.

4.3.3 Calculation of $P_{SC}(X)$

The quality of security of a task t_i with respect to the jth security service is calculated as $\exp(-\lambda_i^j e_i)$, where λ_i^j is a risk rate (see 24) and e_i is the execution time.

$$\lambda_i^j = 1 - \exp\left(-\alpha(1 - s_i^j)\right). \quad (24)$$

Note that this risk rate model assumes that a risk rate is a function of security levels, and the distribution of risk-count for any fixed time interval is approximated using a *Poisson* probability distribution. The risk rate model is for illustration purpose only, and the model can be replaced by any risk rate model with a reasonable parameter α (α is set to 0.002 in our experiments).

The quality of security of t_i can be obtained below by considering all security services. Thus,

$$\prod_{j=1}^{q} \exp\left(-\lambda_i^j\left(e_i + \sum_{l=1}^{q} c_i^l(s_i^l)\right)\right) = \exp\left(-\left(e_i + \sum_{l=1}^{q} c_i^l(s_i^l)\right)\sum_{j=1}^{q} \lambda_i^j\right). \quad (25)$$

Given an allocation X, the probability that all tasks are free from being attacked during the execution of the tasks is computed based on (25). Consequently, we have:

$$P_C(X) = \prod_{i=1}^{n} \exp\left(-\left(e_i + \sum_{l=1}^{q} c_i^l(s_i^l)\right)\sum_{j=1}^{q} \lambda_i^j\right). \quad (26)$$

Likewise, for the kth security service available for a link between M_i and M_j, the quality of security of the link during the time interval t is $\exp(-\hat{\lambda}_{ij}^k t)$, where $\hat{\lambda}_{ij}^k$ denotes the risk rate. Without loss of generality, the risk rate is expressed as the following function of the corresponding security level.

$$\hat{\lambda}_{ij}^k = 1 - \exp\left(-\beta(1 - s_{ij}^k)\right). \quad (27)$$

The quality of security of a message $(t_a, t_b) \in E$ is calculated by taking all security services provided to the message into account. Thus,

$$\prod_{k=1}^{p} \exp(-\hat{\lambda}_{ij}^k \frac{d_{ab}}{B_{ij}}) = \exp\left(-\frac{d_{ab}}{B_{ij}} \sum_{k=1}^{p} \hat{\lambda}_{ij}^k\right), \quad (28)$$

where x_{ai}=1, x_{bj}=1.

Given an allocation X, the probability that all messages allocated to the link between M_i and M_j are risk-free is computed as the product of the quality of security of all the messages. Then, we have:

$$P_{ij}(X) = \prod_{a=1}^{n} \prod_{b=1, b\ne a}^{n} \exp\left[x_{ai} x_{bj}\left(-\frac{d_{ab}}{B_{ij}} \sum_{k=1}^{p} \hat{\lambda}_{ij}^k\right)\right]. \quad (29)$$

Let $P_L(X)$ be the quality of security of all links under allocation X, and $P_L(X)$ can be written as:

$$P_L(X) = \prod_{i=1}^{m} \prod_{j=1, j\ne i}^{m} P_{ij}(X). \quad (30)$$

Finally, the probability $P_{SC}(X)$ can be calculated as follows, where $P_C(X)$ and $P_L(X)$ are obtained from (26) and (30).

$$P_{SC}(X) = P_C(X) P_L(X) \quad (31)$$

4.4 Performance Evaluation

To demonstrate the strength of TAPADS, we compare it with the LIST algorithm, a well-known scheduler for parallel applications. To make the comparisons fair, we slightly modified LIST into three variants: LISTMIN, LISTMAX, and LISTRND, in a way that these schemes can meet parallel applications' security requirements in a heuristic manner. However, these three algorithms make no effort to optimize quality of security. We believe that comparing TAPADS with the three non-security-aware scheduling policies is meaningful because this way the security improvements brought by a security-aware scheduler can be clearly noticed. The three baseline algorithms are follows.

(1) LISTMIN: The scheduler intentionally selects the lowest security level of each security services required by each task of a parallel job.

(2) LISTMAX: The scheduler chooses the highest security level for each security requirement posed by each task within a parallel job.

(3) LISTRND: Unlike the above two baseline algorithms, LISTRND randomly picks a value within the security level range of each service required by a task.

4.4.1 Simulator and Simulation Parameters

Some preliminary results of this part of the study have been presented in [34]. A simulator was designed and implemented based on the model and the algorithm described in the previous sections. Table 3 summarizes the key configuration parameters of the simulated clusters.

The parameters of nodes are chosen to resemble real-world workstations like Sun SPARC-20 and Sun Ultra 10.

TABLE 3. CHARACTERISTICS OF SYSTEM PARAMETERS

Parameter	Value (Fixed) - (Varied)
CPU Speed	100 million instructions/second
Network bandwidth	100 Mbps
Number of nodes	(32, 64,128, 256), (8, 12, 16, 20)
Required security services	Confidentiality, Integrity, and Authentication
Weight of security services	0.2 (authentication), 0.5 (confidentiality), 0.3 (integrity)

The performance metrics by which we evaluate system performance include: *Security Value* (see 12); *QSA:* the quality of security for applications (see 31); *Guarantee factor:* it is zero if a job's deadline cannot be met. Otherwise, it is one; and *Job completion time:* earliest time that a job can finish its execution.

4.4.2 Scalability

This experiment is intended to investigate the scalability of our algorithm. We scale the number of nodes, or PEs, in the cluster from 32 to 256. Note that PE (Processing Element) and node are interchangeable throughout this paper. We used a task graph with 520 tasks, and the deadline is set to 400 seconds. Fig. 3 plots the performances as functions of the number of nodes in the cluster.

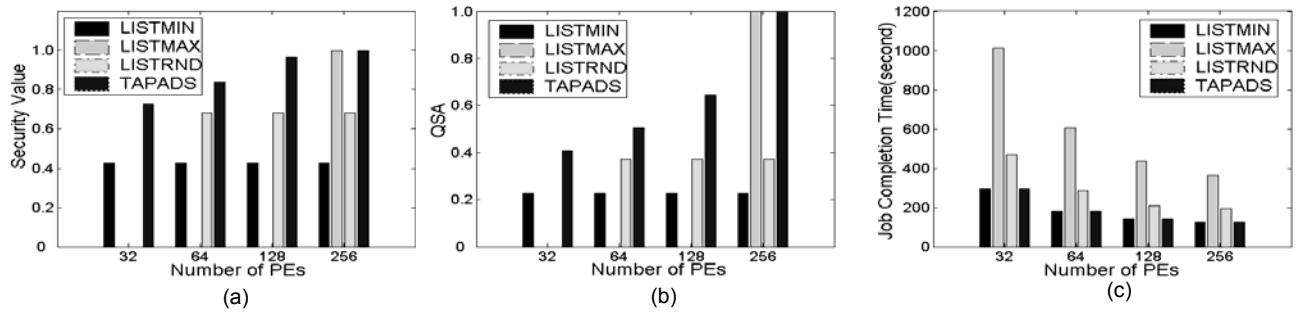


Fig. 3. Performance impact of number of nodes.

The results show that TAPADS exhibits good scalability. It is observed from Fig. 3a that the amount of improvement over LISTMIN becomes more prominent with the increasing value of node number. This result can be explained by the conservative nature of LISTMIN, which merely meets the minimal security requirements for jobs. Conversely, LISTMAX can only achieve the same performance as TAPADS when there are 256 nodes. This is because LISTMAX only guarantees the maximal security requirements of jobs when more nodes are available. We observe from Fig. 3c that all four algorithms can finish the job in a shorter time period when more nodes are available.

4.4.3 Evaluation in Real Application

To validate the results from the synthetic task graphs, we evaluated the TAPADS scheme using a real system – digital signal processing system (DSP) [31]. Fig. 4 shows the impact of deadlines on these schemes, and Fig. 5 reveals the scalability of the four algorithms. Figures 4 and 5

Since there is no widely accepted benchmark graph for the scheduling of parallel jobs represented by DAGs [15], we use random graphs with diverse parameters to test the performance of the TAPADS algorithm. The synthetic parallel job used for Sections 4.4.2 was created by TGFF [7], a randomized task graph generator. We believe that random graphs with different parameters can approximate various types of real-world parallel applications. Section 4.4.3 validates the experimental results from synthetic task graphs by using a real-world application – digital signal processing system (DSP) [31]. In a random task graph, the computational time (security overhead is not included) of each node (task) was randomly selected from a triangular distribution. Similarly, the size of security-required data generated by a task was arbitrarily chosen from a triangular distribution. Based on our observations on practical parallel jobs, the majority of tasks in a parallel job have a very similar execution time, while only a few tasks have either a very short execution time or a quite long execution time. The maximal number of out degrees in a task graph was set to 25. The number of in degrees is randomly selected from a uniform distribution in the range [1, 10].

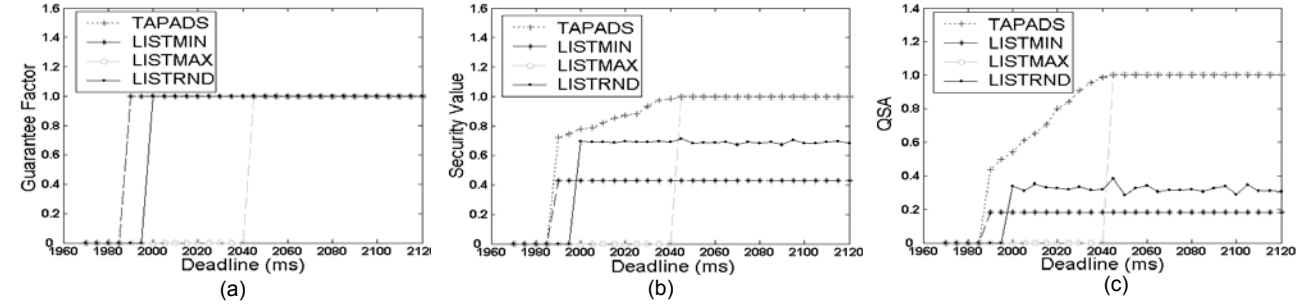


Fig. 4. Performance impact of deadline for DSP.

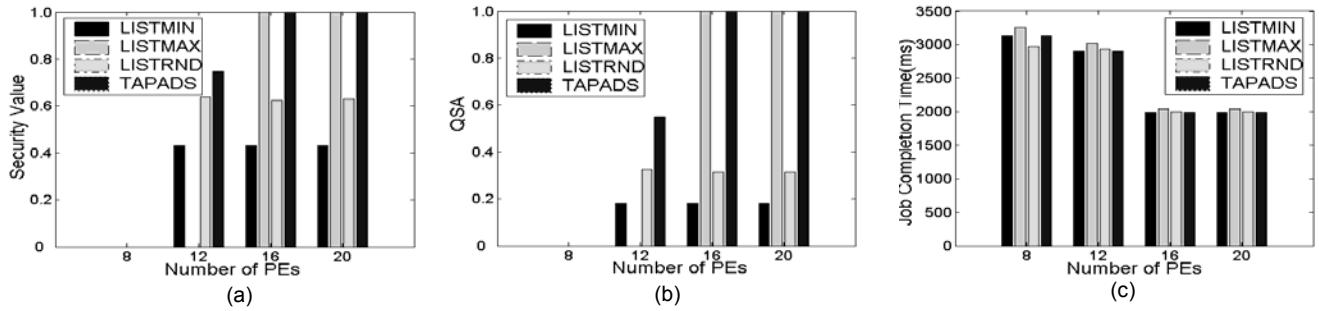


Fig. 5. Performance impact of number of nodes for DSP.

demonstrate that TAPADS can gain performance improvements in a real system. The strength of TAPADS can be fully exhibited when the application has a relatively tight deadline. When the deadline is loose, TAPADS reduces to LISTMAX. It is suggested that TAPADS can significantly improve security value and QSA without increasing hardware cost when applications have tight deadline requirements.

5 Security-Aware Resource Allocation for Heterogeneous Clusters

TAPADS presented in the previous section can significantly improve the performance of homogeneous clusters in terms of security and schedulability. However, the TAPADS scheme has no inherent capability of supporting heterogeneous clusters because it assumes that all nodes in a cluster are identical in terms of computation capacity. This assumption is not always valid in reality. Still, there are many heterogeneous clusters [10][20] on which parallel jobs with real-time and security requirements are running. In a heterogeneous cluster, the computation capacities of computing nodes are diverse. As a result, the execution time of a task t_i in a heterogeneous cluster is a vector of values rather than one fixed value. Similarly, the security overhead of task t_i is also decided by which computing node it is assigned. These two new challenges in the forms of computational heterogeneity and security heterogeneity prevent TAPADS from being applied in heterogeneous clusters. In this regard, we are motivated to introduce the concept of security heterogeneity, and to propose a heterogeneity-aware resource allocation algorithm to improve security of real-time parallel applications running on heterogeneous clusters.

5.1 Modeling Computational Heterogeneity and Security Heterogeneity

We consider a class of embarrassingly parallel applications (see [30] for some examples) each of which can be envisioned as a set of tasks without any interaction between one another. An application is modeled as a tuple *(T, a, f, d, l)*, where $T = \{t_1, t_2, ..., t_n\}$ represents a set of n tasks, a and f are the arrival and finish times, d is the specified deadline, and l denotes the amount of data (measured in MB) to be protected. Each task $t_i \in T$ is labeled with a pair, e.g., $t_i = (E_i, S_i)$, where E_i and S_i are vectors of execution times and security requirements for task t_i. The execution time vector denoted by $E_i = (e_i^1, e_i^2, \ldots, e_i^m)$ represents the execution time of t_i on each node in the cluster. Each task of a parallel application requires a set of security services providing various security levels, which are normalized in the range from 0.1 to 1.0. Suppose $t_i \in T$ requires q security services, $S_i = (s_i^1, s_i^2, \ldots, s_i^q)$, a vector of security levels, characterizes the security requirements of the task. The impacts of these two heterogeneities on system performance and security will be investigated in Section 5.4.3.

Let w_i^j denote the computational weight of task t_i on node m_j. w_i^j is computed as a ratio between its execution time on m_j and that on the fastest node in the cluster. The computational heterogeneity level of t_i, referred to as H_i^C, is quantitatively measured by the standard deviation of the computational weights. That is, H_i^C is expressed as:

$$H_i^C = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(w_i^{avg} - w_i^j\right)^2}, \quad (32)$$

where $w_i^j = e_i^j \Big/ \min_{k=1}^{n}\left(e_i^k\right)$ and $w_i^{avg} = \left(\sum_{j=1}^{n} w_i^j\right) \Big/ n$.

The computational heterogeneity of a parallel application with task set T is calculated as:

$$H^C = \frac{1}{|T|}\sum_{T_i \in T} H_i^C \quad (33)$$

Besides computation heterogeneity, a cluster may exhibit security heterogeneity. Each node provides an array of security services measured by security levels normalized in the range from 0.1 to 1.0. Security services provided by node m_j is characterized as a vector of security levels, $P_j = (p_j^1, p_j^2, \ldots, p_j^q)$, where p_j^k $(1 \le k \le q)$ is the security level of the kth security service provided by m_j.

Given a task t_i and its security requirement $S_i = (s_i^1, s_i^2, \ldots, s_i^q)$, the heterogeneity of security requirement for t_i is represented by the standard deviation of the security levels in the vector. Thus,

$$H_i^S = \sqrt{\frac{1}{q}\sum_{j=1}^{q}\left(s_i^{avg} - s_i^j\right)^2}, \quad (34)$$

where $s_i^{avg} = \left(\sum_{j=1}^{q} s_i^j\right) \Big/ n$.

The security requirement heterogeneity of a parallel application with task set T is computed by:

$$H^S = \frac{1}{|T|}\sum_{T_i \in T} H_i^S \quad (35)$$

The heterogeneity of the kth security service in a heterogeneous cluster is expressed as:

$$H_k^V = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(p_{avg}^k - p_i^k\right)^2}, \quad (36)$$

where $p_{avg}^k = \left(\sum_{i=1}^{n} p_i^k\right) \Big/ n$.

Similarly, the heterogeneity of security services in node m_j is expressed as:

$$H_j^M = \sqrt{\frac{1}{q}\sum_{k=1}^{q}\left(p_j^{avg} - p_j^k\right)^2}, \quad (37)$$

where $p_j^{avg} = \left(\sum_{k=1}^{q} p_j^k\right) \Big/ q$.

Using (36), the heterogeneity in security services of the cluster can be computed as:

$$H^V = \frac{1}{|q|}\sum_{k=1}^{q} H_k^V. \quad (38)$$

Now we consider heterogeneity in security overhead. As before, the following model account for three security services, including confidentiality, integrity, and authentication [33]. Let s_i^k and $c_{ij}^k(s_i^k)$ be the security level and overhead of the kth security service, the security overhead c_{ij} experienced by t_i on node m_j can be computed using (39) and (40).

$$c_{ij} = \sum_{k=1}^{q} c_{ij}^k(s_i^k), \text{ where } s_i^k \in S_i. \quad (39)$$

$$c_{ij} = \sum_{k\in\{a,e,g\}} c_{ij}^k(s_i^k), \text{ where } s_i^k \in S_i. \quad (40)$$

where $c_{ij}^e(s_i^e)$, $c_{ij}^g(s_i^g)$, and $c_{ij}^a(s_i^a)$ are overheads caused by the authentication, confidentiality, and integrity services. Finally, the security overhead of a task set T is calculated by:

$$c = \sum_{t_i\in T}\sum_{j=1}^{n} x_{ij} c_{ij} = \sum_{t_i\in T}\sum_{j=1}^{n}\left(x_{ij}\sum_{k\in\{a,e,g\}} c_{ij}^k(s_i^k)\right), \quad (41)$$

where $x_{ij} = 1$ if t_i is allocated to node m_j, $\sum_{j=1}^{n} x_{ij} = 1$, and $s_i^k \in S_i$.

5.2 The SHARP Algorithm

5.2.1 Problem Formulation

We introduce a closed form expression for the security benefit of task t_i. Thus, the security benefit of t_i is measured by *Security Deficiency* (SD), which is quantified as the discrepancy between requested security levels and offered security levels. The SD value of the kth service is defined as:

$$g(s_i^k, p_j^k) = \begin{cases} 0, \text{ if } s_i^k \le p_j^k \\ s_i^k - p_j^k, \text{ otherwise} \end{cases}, \quad (42)$$

where t_i is allocated to m_j.

For the kth security service, a small SD value indicates a high degree of service satisfaction. A zero SD value implies that t_i's requirement placed on the kth security service can be perfectly met. The SD value of t_i on m_j can be derived from (43). Thus, the SD value of t_i is computed as a weighted sum of the SD values of q required security services. Formally, we have:

$$SD(s_i, P_j) = \sum_{k=1}^{q}\left[w_i^k g\left(s_i^k, p_j^k\right)\right], \quad (43)$$

where w_i^k is the weight of the kth security service, $0 \le w_i^k \le 1$, and $\sum_{k=1}^{q} w_i^k = 1$.

Likewise, the security benefit of a parallel application with task set T is measured by *Degree of Security Deficiency* (DSD), which is defined as the sum of the security deficiency values of all the tasks in the task set. Consequently, the DSD value of task set T under allocation X can be written as:

$$DSD(T, X) = \sum_{t_i\in T}\sum_{j=1}^{n}\left[x_{ij} SD(s_i, P_j)\right] = \sum_{t_i\in T}\sum_{j=1}^{n}\left\{x_{ij}\sum_{k=1}^{q}\left[w_i^k g\left(s_i^k, p_j^k\right)\right]\right\}, \quad (44)$$

where $x_{ij} = 1$ if t_i is allocated to node m_j, $\sum_{j=1}^{n} x_{ij} = 1$, and $s_i^k \in S_i$.

Let X be the schedule for all the tasks in task set T. The following objective function needs to be minimized, meaning that the degree of security deficiency of T is optimized.

$$\text{minimize } DSD(T, X) = \sum_{t_i\in T}\sum_{j=1}^{n}\left\{x_{ij}\sum_{k=1}^{q}\left[w_i^k g\left(s_i^k, p_j^k\right)\right]\right\}, \quad (45)$$

subject to $f_i \le d$ and $\forall x_{ij} \in X, \sum_{j=1}^{n} x_{ij} = 1 : x_{ij} = 1$ if t_i is allocated to node m_j, where f_i is the finish time of the ith task in the task set.

Given a heterogeneous cluster and a sequence of submitted parallel applications, the SHARP algorithm is intended to minimize the cluster's overall DSD value defined as the sum of the degree of security deficiency of all the submitted applications. Finally, we can obtain the following non-linear optimization problem formulation for the heterogeneous cluster, subject to the timing constraints:

$$\text{minimize} \sum_{\text{for all T}} DSD(T, X). \quad (46)$$

Thus, SHARP is designed to minimize the average degree of security deficiency.

$$DSD = \sum_{\text{for all T}}\left\{P_{sd}(T)\sum_{t_i\in T}\sum_{j=1}^{n}\left[x_{ij}\sum_{k=1}^{q}\left[w_i^k g\left(s_i^k, p_j^k\right)\right]\right]\right\} \Big/ \sum_{\text{for all T}} P_{sd}(T) \quad (47)$$

where $P_{sd}(T)$ is a step function, and $P_{sd}(T) = \begin{cases} 1, \text{if task set } T \text{ can be timely completed} \\ 0, \text{otherwise} \end{cases}$.

The earliest start time σ_i^j can be computed as:

$$\sigma_i^j = \tau + \sum_{t_l\in m.d_l\le d}\left(e_l^j + \sum_{k=1}^{q} c_l^k(\bar{s}_l^k)\right) \quad (48)$$

where τ is the current time, and $\sum_{t_l\in m.d_l\le d}\left(e_l^j + \sum_{k=1}^{q} c_l^k(\bar{s}_l^k)\right)$ is the overall execution time (security overhead is factored in) of all tasks with earlier deadlines than d. If task t_i is running on node m_j, the start time σ_i^j is the earliest available time of t_i on m_j.

5.2.2 Algorithm Description

The SHARP algorithm is outlined in Fig. 6. The goal of the algorithm is to deliver high quality of security under two conditions: (1) deadlines of submitted parallel applications are met; and (2) the degree of security deficiency (see 44) of each admitted parallel application is minimized.

1. Select a parallel application, which has the earliest deadline among applications in the arrival queue;
2. **for** each task t_i of the application chosen in step 1 **do**
3. Initialize the security deficiency of task t_i, $SD_i \leftarrow \infty$;
4. **for** each node m_j in the heterogeneous cluster **do**
5. Use (48) to compute σ_i^j
6. Calculate t_i's security overhead $\sum_{k=1}^{q} c_{ij}^k(\bar{s}_i^k)$, where $\bar{s}_i^k = \begin{cases} s_i^k, \text{if } s_i^k \le p_j^k \text{; (see 39)} \\ p_j^k, \text{otherwise} \end{cases}$
7. **if** $\sigma_i^j + e_i^j + \sum_{k=1}^{q} c_{ij}^k(\bar{s}_i^k) \le d$ **then**
8. Use (43) to compute the security deficiency $SD(s_i, P_j)$ of task t_i on node m_j;
9. **if** $SD(s_i, P_j) < SD_i$ **then**
10. $SD_i \leftarrow SD(s_i, P_j)$;
11. $x_{ij} \leftarrow 1$; $\forall k \neq j$: $x_{ik} \leftarrow 0$;
12. **end if**
13. **end if**
14. **end for**
15. **if** no feasible schedule is available for t_i **then**
16. Reject the parallel application;
17. **else**
18. Record the start time and the finish time of task t_i on node m_j where x_{ij}=1;
19. Update the schedule on node m_j where x_{ij}=1;
20. **end if**
21. **end for**
22. **if** all the tasks of the parallel application can be finished before deadline d **then**
23. **for** each task t_i of the parallel application **do**
24. allocate task t_i to node m_j, subject to $1 \le j \le n$, $x_{ij} = 1$;
25. **end for**
26. **end if**

Fig. 6. The SHARP algorithm.

Before reducing the security deficiency value of each task of a parallel application, SHARP makes an effort to meet the timing constraint of the application. This can be accomplished by calculating the earliest start time and the security overhead of each task (see 39) in Steps 5 and 6, followed by checking if all the tasks of the application can be completed before its deadline d (see Step 7). If the deadline of a task in the application can not be met, the application is rejected by Step 16.

The security deficiency value of each task in the application is minimized in the following way. Step 7 is intended to identify a set of candidate nodes satisfying the timing constraint. Steps 9-11 are used to choose a node with the minimal security deficiency among the candidate nodes. Thus, SHARP eventually allocates each task to a node that can reduce security deficiency while meeting the real-time requirement of parallel applications.

Theorem 2. *The time complexity of SHARP is O(mnq), where m is the number of nodes in a cluster, n is the number of tasks in a parallel application, and q is the number of security services.*

Proof. Selecting a parallel application with the earliest deadline takes constant time *O(1)*. The time complexity of finding the security overhead of each task on a node is *O(q)* (Step 6), since SHARP considers q security services. The time complexity of feasibility checking is a constant *O(1)* (Step 7). Since there exist m nodes and n tasks, Steps 5-13 are executed for mn times. Therefore, the time complexity of Steps 2-17 is bounded by *O(mnq)*. Steps 18-22 take O(n) time to allocate n task to m nodes in the cluster. Thus, the time complexity of SHARP is *O(1+nmq+n) = O(nmq)*. □

5.3 Evaluation of Security Risks

Now we derive the probability $P_{rf}(t_i,m_j)$ that t_i remains risk-free during the course of its execution on node m_j. It is to be noted that the risk-free probability can be used as a complementary means of quantifying the quality of security. The risk-free probability of task t_i with respect to the kth service is:

$$P_{rf}^k(t_i,m_j) = \exp\left(-\lambda_i^k\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l(s_i^l)\right)\right), \quad (49)$$

where λ_i^k is the risk rate (see 24), and $c_{ij}^l(s_i^l)$ is the security overhead.

The risk-free probability of task t_i on node m_j can be written as (50), where all security services provided to the task are considered. Thus, we have:

$$\begin{aligned} P_{rf}(t_i,m_j) &= \prod_{k=1}^{q} P_{rf}^k(t_i,m_j) \\ &= \prod_{k=1}^{q} \exp\left(-\lambda_i^k\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l(s_i^l)\right)\right) \\ &= \exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l(s_i^l)\right)\sum_{k=1}^{q}\lambda_i^k\right). \end{aligned} \quad (50)$$

Using (50), we can write the overall risk-free probability of task t_i in the cluster as:

$$\begin{aligned} P_{rf}(t_i) &= \sum_{j=1}^{n}\left\{P[x_{ij}=1]\cdot P_{rf}(t_i,m_j)\right\}, \\ &= \sum_{j=1}^{n}\left\{p_{ij}\cdot\exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l(s_i^l)\right)\sum_{k=1}^{q}\lambda_i^k\right)\right\} \end{aligned} \quad (51)$$

where p_{ij} is the probability that t_i is allocated to node m_j, and $p_{ij} = \begin{cases} 1, \text{if } t_i \text{ is assigned onto node } m_j \\ 0, \text{otherwise} \end{cases}$. Given a parallel application with task set T, the probability that all tasks are free from being attacked during their executions is computed based on (51). Consequently, the risk-free probability of the task set can be computed as below:

$$\begin{aligned} P_{rf}(T) &= \prod_{t_i\in T} P_{rf}(t_i) \\ &= \prod_{T_i\in T}\left\{\sum_{j=1}^{n}\left\{p_{ij}\cdot\exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l(s_i^l)\right)\sum_{k=1}^{q}\lambda_i^k\right)\right\}\right\}. \end{aligned} \quad (52)$$

Finally, we can calculate the average risk-free probability of all schedulable parallel applications on a heterogeneous cluster using (53), where $P_{sd}(T)$ is a step function, and $P_{sd}(T) = \begin{cases} 1, \text{if } T \text{ can be timely completed} \\ 0, \text{otherwise} \end{cases}$. It is

worth noting that the SHARP approach is conducive to maximizing the risk-free probabilities of heterogeneous clusters. As we explained in Section 5.2.2, for each task t_i, Steps 9-11 of SHARP (see Fig. 9) choose a node with the minimal security deficiency among the candidate nodes. Therefore, a node who can best meet t_i's security level requirements will be selected by SHARP as the destination node for t_i (see 42). Consequently, t_i will be executed with its required security levels or with higher security levels close to its requirements. In other words, the obtained security levels of t_i will be maximized. Based on (24), a higher security level for the jth security service implies a lower risk rate λ_i^j. A lower risk rate indicates a higher risk-free probability (see 49, 50, and 51), which in turn results in a higher risk-free probability of a heterogeneous cluster (see 53). Thus, SHARP maximizes P_{rf}.

$$P_{rf} = \left(\sum_{\text{for all T}} P_{sd}(T) \cdot P_{rf}(T) \right) \Big/ \sum_{\text{for all T}} P_{sd}(T) = \left\{ \sum_{\text{for all T}} \left[P_{sd}(T) \prod_{T_i \in T} P_{rf}(T_i) \right] \right\} \Big/ \sum_{\text{for all T}} P_{sd}(T) \quad (53)$$

The risk-free probability computed by (53) is used in concert with the degree of security deficiency (see 47) to measure the quality of security provided by a heterogeneous cluster. In the subsequent section, we quantitatively evaluate the risk-free probability and degree of security deficiency for heterogeneous cluster under a wide range of workload conditions.

TABLE 4. SYSTEM PARAMETERS

Parameter	Value (Fixed) - (Varied)
Nnnumber of tasks	(6400) – The first three month trace data from SDSC SP2 log
Number of nodes	(64) – (32, 64, 128, 192, 256)
CPU Speedup	(1) – (2, 3, 4, 5, 6, 7, 8, 9, 10)
Size of security-sensitive data	(1–100) – (0.01–1,0.1–10,1–100,10–1000,100–10000,1000–100000) MB
Computational heterogeneity	(1.08) – (0, 0.43, 1.08, 1.68, 2.27) (see 33)
Security heterogeneity	(0.22) – (0, 0.14, 0.22, 0.34, 0.56) (see 38)

5.4 Performance Results and Comparisons

In purpose of revealing the strength of SHARP, we compared it with two well-known algorithms, namely, EDF (Earliest Deadline First) and LLF (Least Laxity First). These algorithms briefly described below are representative dynamic scheduling algorithms for clusters.

(1) EDF: An algorithm that schedules a ready job with the earliest deadline.

(2) LLF: A heuristic that assigns priority based on laxity of jobs. Job with minimum laxity is assigned highest priority. Laxity = Deadline – Worst case computation time.

Table 4 summarizes the key configuration parameters of the simulated heterogeneous cluster.

5.4.1 Simulation Parameters

The parameters of nodes in the simulated cluster are chosen to resemble real-world workstations like IBM SP2 nodes. We made use of a real world trace (e.g., San Diego Supercomputer Center SP2 log sampled on a 128-node cluster) to conduct simulations. We modified the trace by adding a block of security-sensitive data for each task. "job number", "submit time", "execution time" and "number of requested processors" of jobs submitted to the system are taken directly from the trace. "deadlines", "security requirements of jobs", and "security-sensitive data size" are synthetically generated, since these parameters are not available in the trace. The performance metrics we used include: *Average risk-free probability* (ARFP, see 53), *Average degree of security deficiency* (ADSD, see 47), and *Guarantee ratio* (GR, measured as a fraction of total submitted parallel applications that are found to be schedulable). While ADSD gives users a quantitative way to compare different scheduling algorithms in terms of their security service satisfaction abilities, ARFP provides us a means of measuring probabilities of risk-free task executions supplied by distinct scheduling schemes. Although both ADSD and ARFP are security-related performance metrics, they complement each other by offering two different angles to evaluate the quality of security delivered by scheduling algorithms. GR is a traditional performance metric to evaluate scheduling algorithms. A high performance scheduling algorithm can result in a high value of GR, which means the majority of submitted jobs can be scheduled so that their deadlines are met.

5.4.2 Impact of the Size of Security-Sensitive Data

In this set of experiments we evaluated the performance impact of security-sensitive data size. We tested six configurations of size of data to be secured (see Table 4).

The experimental results are shown in Fig. 7 When the security-sensitive data size increases, the degree of security deficiency of SHARP slightly increases. This observation can be explained as follows. When SHARP is de-

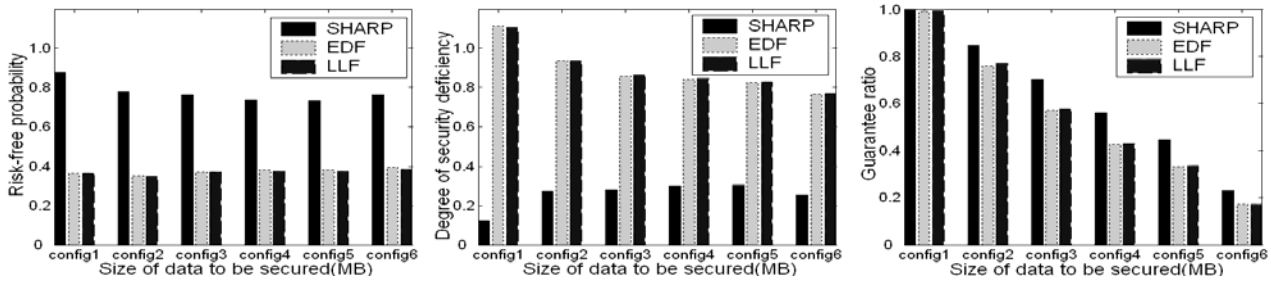


Fig. 7. Performance impact of size of data to be secured.

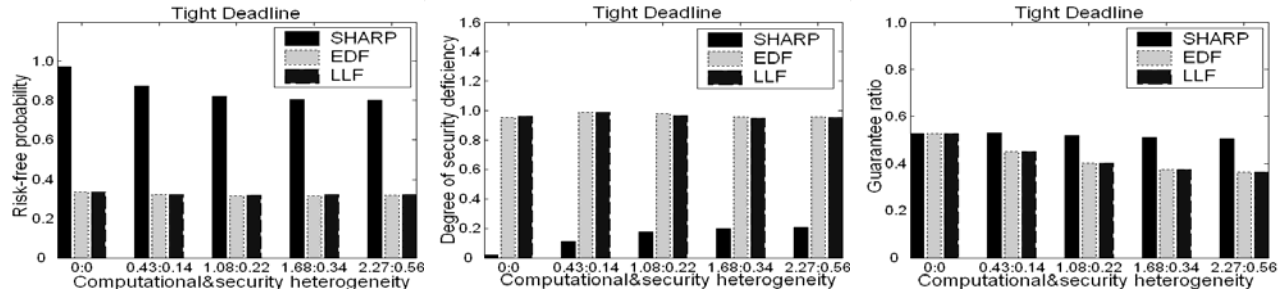


Fig. 8. Performance impact of security and computational heterogeneities.

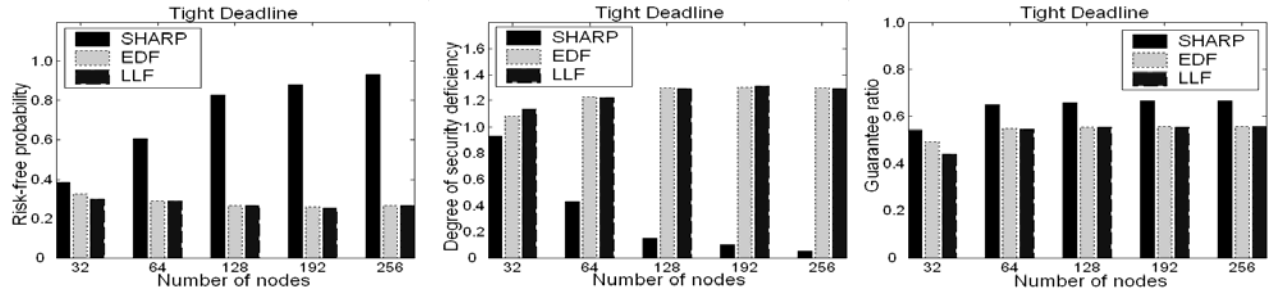


Fig. 9. Performance impact of number of nodes.

ployed in the cluster, security overhead becomes moderately dominant with the growing data size and, therefore, tasks of a parallel application are more likely to be allocated to nodes providing lower security levels. The nodes with small total execution time have low security overhead, meaning that security levels offered by these nodes are lower. Thus, the degree of security deficiency for SHARP enlarges with increasing data size. Unlike SHARP, the degrees of security deficiency of EDF and LLF marginally reduce with the increasing value of data size. This result is reasonable because EDF and LLF only admit applications with low security demands when security-sensitive data size is large, thereby being able to meet the security constraints of most admitted applications.

5.4.3 Heterogeneities in Security and Computation

In this experiment, we investigate the performance impacts of heterogeneities in security and computation. The five heterogeneity configurations are detailed in Table 4.

Fig. 8 shows that SHARP fully exhibits its strength when the heterogeneities increase, e.g., SHARP substantially performs better than the alternatives. Additionally, the risk-free probabilities and degrees of security deficiency of EDF and LLF marginally change when the security and computational heterogeneities increase. When deadlines are tight, SHARP is significantly superior to EDF and LLF in terms of guarantee ratio. The implication behind this result is that SHARP is the most appropriate algorithm for scenarios where parallel applications on heterogeneous clusters have tight deadlines.

5.4.4 Scalability

This group of experiments is intended to investigate the scalability of SHARP. We scale the number of nodes in a heterogeneous cluster from 32 up to 256. It is observed from Fig. 9 that SHARP makes more prominent improvement in degree of security deficiency and risk-free probability when the heterogeneous cluster size scales up. Importantly, SHARP can achieve high performance provided that there exist a large number of nodes in the cluster, because there is a strong likelihood that SHARP can meet applications' security demands while minimizing the execution times.

5.4.5 CPU Capacity

In this set of experiments we examine security and performance sensitivities of the three algorithms to CPU capacities. We varied the CPU capacity (measured as speedup over the baseline computational node) from 2 to 10. The CPU speed of the IBM SP2 66MHz nodes is normalized to 1. We normalized the CPU capacity of the nodes to the values from 2 to 10. The laxity is set to 10000 seconds, and the number of nodes is fixed to 32.

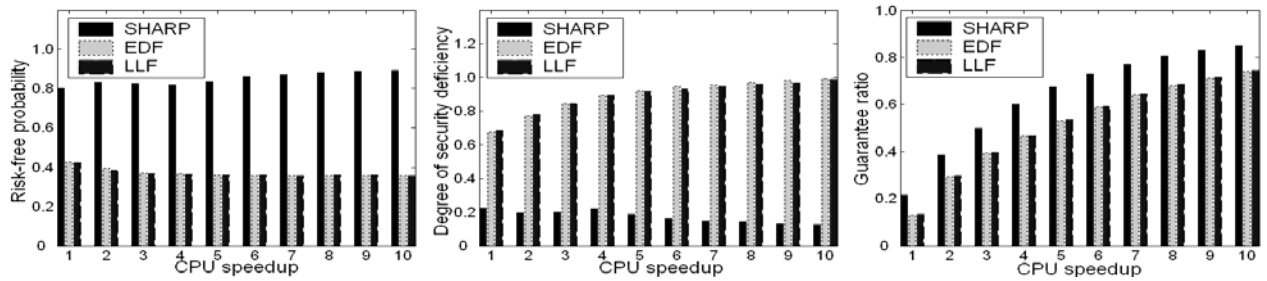


Fig. 10. Performance impact of CPU speedup.

As before, Fig. 10 reveals that SHARP is superior to the other two competitors in all the three performance metrics. In addition, the improvements of SHARP in degree of security deficiency and risk-free probability become more prominent when the CPU capacity increases. These results as well as those presented in Section 5.4.4 indicate that SHARP exhibits good scalability, and SHARP can improve both security and performance of large-scale heterogeneous clusters with powerful CPUs.

6 Conclusions

This paper aims at presenting security-aware resource allocation schemes for real-time parallel applications running on clusters. The schemes consider two parallel application models where timeliness and security requirements are factored in. For the first part of this study, we propose TAPADS, an allocation scheme makes use of critical path analysis as well as security level refinement to maximize security and schedulability. In the second part of the study, we develop SHARP, a security-aware resource allocation algorithm for real-time jobs on heterogeneous clusters. SHARP is applied to maximize the probability that parallel applications are timely executed without any risk of being attacked.

Future studies can be performed in the following directions. First, we will extend the heuristic schemes to accommodate data transmissions among disk I/O nodes. Second, we will propose a security-aware resource allocation scheme where multi-dimensional computing resources are considered. For now we only consider CPU time, which is one of the computing resources consumed by security services. Still, security services require other resources like memory, network bandwidth and storage capacities. They might compete with submitted parallel jobs for these resources. As a result, the resource competition could noticeably affect the computation time of both submitted jobs and their required security services. We will investigate the impact of resource competition on computation time in our future work. Finally, we intend to incorporate more security services (e.g., authorization and auditing services) into our resource allocation schemes.

Acknowledgment

The work reported in this paper was supported by the US National Science Foundation under Grant No. CCF-0742187, San Diego State University under a startup fund, Auburn University under a startup grant, the Intel Corporation under Grant No. 2005-04-070, and the Altera Corporation under an equipment grant. The authors wish to thank the anonymous reviewers for their helpful comments.

References

[1] T.F. Abdelzaher, E. M. Atkins, and K.G. Shin., "QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control," *IEEE Trans. Computers*, vol. 49, pp.1170-1183, 2000.

[2] Q. Ahmed and S. Vrbsky, "Maintaining security in firm real-time database systems," *Proc. 14th Ann. Computer Security Application Conf.*, pp. 83-90, 1998.

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

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

[5] R. Chandramouli, S. Bapatla, K. P. Subbalakshmi and R. N. Uma, "Battery power-aware encryption," *ACM Trans. Information and System Security*, vol. 9 , no. 2, pp. 162-180, 2006.

[6] K. Connelly and A. A. Chien, "Breaking the barriers: high performance security for high performance computing," *Proc. Workshop on New Security Paradigms*, pp. 36-42, 2002.

[7] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," *Proc. Int. Workshop. Hard-ware/Software Codesign*, pp. 97-101, 1998.

[8] I. Foster, N.T. Karonis, C. Kesselman and S. Tuecke, "Managing security in high-performance distributed computations," *Cluster Computing*, vol. 1, no. 1, pp. 95-107, 1998.

[9] S. Gritzalis, "Enhancing Privacy and Data Protection in Electronic Medical Environments," *Journal of Medical Systems*, vol. 28, no. 6, pp. 535 – 547, 2004.

[10] 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, 2003.

[11] A. Jones and J.C. Rabelo, "Survey of Job Shop Scheduling Techniques," *NISTIR*, National Institute of Standards and Technology, 1998.

[12] C.-J. Hou and K. G. Shin, "Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems," *IEEE Trans. Computers*, vol. 46, no. 12, pp.1338-1356, 1997.

[13] W.T.C. Kramer, A. Shoshani, D.A. Agarwal, B.R. Draney, G. Jin, G.F. Butler, and J.A. Hules, "Deep scientific computing requires deep data," *IBM Journal of Research and Development*, vol. 48, no. 2, pp. 209 – 232, 2004.

[14] B. Krebs, "Hackers Strike Advanced Computing Networks," *Washington Post*, April 2004.

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

[16] C. Irvine and T. Levin, "Towards a taxonomy and costing method for security services," *Proc. 15th Ann. Computer Security Applications Conf.*, pp. 183-188, 1999.

[17] W. Li and R.B. Vaughn, "Cluster Security Research Involving the Modeling of Network Exploitations Using Exploitation Graphs," *Proc. Sixth IEEE Int'l Symp. Cluster Computing and Grid*, pp. 26-36, 2006.

[18] B. Littlewood, S. Brocklehurst, N. E. Fenton, P. Mellor, S. Page, D. Wright, J. Dobson, J. McDermid and D. Gollmann, "Towards Operational Measures of Computer Security," *Journal of Computer Security*, vol. 2, no. 3, pp. 211-230. 1993.

[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] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of inde-

pendent tasks onto heterogeneous computing systems," *Proc. IEEE Heterogeneous Computing Workshop*, pp. 30–44, 1999.

[21] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti and E. Roback, "Status Report On The First Round Of The Development Of The Advanced Encryption Standard," *Journal of the Research of the National Institute of Standards and Technology*, vol. 104, no. 5, pp. 435-459, 1999.

[22] J. Pamula, S. Jajodia, P. Ammann and V. Swarup, "A weakest-adversary security metric for network configuration security analysis," *Proc. 2nd ACM workshop on Quality of Protection*, pp. 31-38, 2006.

[23] M. Pourzandi, D. Gordon, W. Yurcik and G. A. Koenig, "Clusters and security: distributed security for distributed systems," *Proc. Fifth IEEE Int'l Symp. Cluster Computing and the Grid*, pp. 96-104, 2005.

[24] W. Shi, H.H.S. Lee, C. Lu and M. Ghosh, "Towards the issues in architectural support for protection of software execution," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 1, *Special issue workshop on architectural support for security and anti-virus (WASSA)*, pp. 6-15, 2005.

[25] S. Song, K. Hwang, Y.K. Kwok, "Risk-Resilient Heuristics and Genetic Algorithms for Security-Assured Grid Job Scheduling," *IEEE Trans. Computers*, Vol. 55, No. 6, June 2006.

[26] J. A. Stankovic, M. Spuri, K. Ramamritham, G.C. buttazzo, "Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms," *Kluwer Academic Publishers*, ISBN 978-0792382690, 1998.

[27] 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, 2002.

[28] 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.

[29] A. Wagner, H.-W. Jin, D.K. Panda, and R. Riesen, "NIC-based Offload of Dynamic User-defined Modules for Myrinet Clusters," *IEEE Int'l Conf. Cluster Computing*, pp. 205 – 214, 2004.

[30] B. Wilkinson and M. Allen, "Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers", Prentice-Hall, Inc., ISBN 978-0136717102, 1999.

[31] C.M. Woodside and G.G. Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems", *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 2, pp. 164-174, 1993.

[32] T. Xie, "Security-Aware Scheduling for Real-Time Systems", *Ph.D. Dissertation*, New Mexico Institute of Mining and Technology, May 2006.

[33] T. Xie and X. Qin, "Scheduling Security-Critical Real-Time Applications on Clusters," *IEEE Trans. Computers*, vol. 55, no. 7, pp. 864-879, 2006.

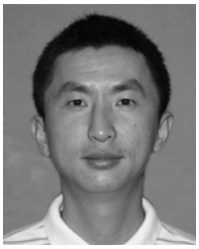
[34] T. Xie and X. Qin, "A New Allocation Scheme for Parallel Applications with Deadline and Security Constraints on Clusters," *The 7th IEEE Int'l Conf. on Cluster Computing*, pp. 1-10, 2005.

[35] T. Xie and X. Qin, "A Security Middleware Model for Real-time Applications on Grids," *IEICE Transactions on Information and Systems, Special Issue on Parallel/Distributed Computing and Networking*, Vol.E89-D, No.2, pp.631-638, 2006.

[36] T. Xie and X. Qin, "A Security-Oriented Task Scheduler for Heterogeneous Distributed Systems," *Proc. 13th Annual IEEE Int'l Conf. High Performance Computing*, *Lecture Notes in Computer Science (LNCS 4297)*, ISSN 0302-9743, pp. 35-46, 2006.

[37] T. Xie, X. Qin and M. Nijim, "SHARP: A New Real-Time Scheduling Algorithm to Improve Security of Parallel Applications on Heterogeneous Clusters," *Proc. 25th IEEE Int'l Performance Computing and Communications Conf.*, April 10-12, 2006.

[38] Q. Zheng and K.G. Shin, "On the Ability of Establishing Real-Time Channels in Point-to-Point Packet Switched Network," *IEEE Trans. Comm.*, vol. 42, pp. 1096-1105, 1994.



Tao Xie received the PhD degree in computer science from the New Mexico Institute of Mining and Technology in 2006. He received the BSc and MSc degrees from Hefei University of Technology, China, in 1991 and 2000, respectively. He is currently an assistant professor in the Department of Computer Science at San Diego State University, San Diego, California. His research interests are security-aware scheduling, high performance computing, cluster and Grid computing, parallel and distributed systems, real-time/embedded systems, storage systems, and information security. He is a member of the IEEE.

Xiao Qin (S'99-M'04) received the BS and MS degrees in Computer Science from Huazhong University of Science and Technology, China, in 1996 and 1999, respectively. He received the PhD in Computer Science from the University of Nebraska-Lincoln in 2004. Currently, he is an Assistant Professor of Computer Science at Auburn University. Prior to joining Auburn University in 2007, he had been with New Mexico Institute of Mining and Technology for three years. In 2007, he received an NSF Computing Processes & Artifacts (CPA) Award. His research interests include parallel and distributed systems, real-time computing, storage systems, and fault tolerance. He has been on the program committees of various international conferences, including IEEE Cluster, IEEE IPCCC, and ICPP. He had served as a subject area editor of IEEE Distributed System Online (2000-2001). He is a member of the IEEE.