

SHARP: A New Real-Time Scheduling Algorithm to Improve Security of Parallel Applications on Heterogeneous Clusters

Tao Xie, Xiao Qin[‡], and Mais Nijim

Department of Computer Science
New Mexico Institute of Mining and Technology
Socorro, New Mexico 87801
(xietao,xqin,mais)[‡]@cs.nmt.edu

Abstract

This paper addresses the problem of improving quality of security for real-time parallel applications on heterogeneous clusters. We propose a new security- and heterogeneity-driven scheduling algorithm (SHARP for short), which strives to maximize the probability that parallel applications are executed in time without any risk of being attacked. Because of high security overhead in existing clusters, an important step in scheduling is to guarantee jobs' security requirements while minimizing overall execution times. The SHARP algorithm accounts for security constraints in addition to different processing capabilities of each node in a cluster. We introduce two novel performance metrics, degree of security deficiency and risk-free probability, to quantitatively measure quality of security provided by a heterogeneous cluster. Both security and performance of SHARP are compared with two well-known scheduling algorithms. Extensive experimental studies using real-world traces confirm that the proposed SHARP algorithm significantly improves security and performance of parallel applications on heterogeneous clusters.

1 Introduction

A heterogeneous cluster consists of an array of diverse computers, called nodes, which are connected by a high-performance network. To date heterogeneous clusters have been emerging as popular computing platforms for computationally intensive applications with diverse computing needs [5]. Recently there have been some efforts devoted to development of real-time parallel applications on clusters [1] [10] [11]. Real-time parallel applications depend not only

on results of computation, but also on time instants at which these results become available.

Today there exist a growing number of parallel applications demanding both high security and real-time performance, because sensitive data and parallel processing require special safeguard and protection against unauthorized access. More importantly, security becomes critical for a wide range of real-time applications on heterogeneous clusters [1][2][4]. For example, in a real-time stock quote update and trading system, each incoming request from a business partner and each outgoing response from an enterprise's back-end application have deadlines and security requirements, which have to be processed by a cluster located between the business partners and enterprise back-end applications [6]. The security requirements here could be "Routing + message security" or "Routing + SSL + client authentication" and so on [6]. Unfortunately, heterogeneous clusters are constructed to execute a broad spectrum of user-compiled applications from a great number of different users. Both applications and users can be sources of security threats to clusters [18]. For instance, the vulnerabilities of applications can be exploited by hackers to compromise the clusters, and malicious users can access the clusters to launch a variety of attacks such as denial of service (DOS) attacks. Even a legitimate user may accidentally tamper with shared data or excessively consume computing cycles to disrupt services available to other cluster users [18]. Thus, it is mandatory to deploy security services to protect security-critical applications running on clusters. The three most common attacks in heterogeneous cluster environments are snooping, alteration, and spoofing. To counter the attacks, we considered three corresponding security services (authentication service, integrity service, and confidentiality service) to be deployed in clusters. Snooping is an unauthorized interception of

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

[‡]This paper appeared in the Proceedings of the 25th IEEE International Performance, Computing, and Communications Conference (IPCCC'06), Phoenix, Arizona, April 2006.

information and it 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 [3]. 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 parallel systems like heterogeneous clusters [7][8][15]. However, existing scheduling algorithms perform poorly for security-sensitive and real-time parallel applications due to the oversight and ignorance of security requirements imposed by real-time parallel applications.

Very recently, Song et al. proposed security-driven scheduling algorithms for grids [12]. This study is by far the closest to the proposed algorithm found in the literature. The main difference between our study and theirs are six-fold. (1) Their algorithms are unable to efficiently support real-time applications, whereas our algorithm is designed for parallel applications with timing constraints. (2) Their study was focused on grids instead of clusters. (3) Their algorithms did not explicitly support multiple security services; rather, it was assumed in their algorithms each grid site could only offer one conceptual security level. Unlike theirs, our scheduling algorithm factors in practical security services with security levels that capture the essence of quality of security. (4) Our algorithm takes into account of heterogeneities in security and computation while theirs only considered homogeneous computing resources. (5) Their algorithms made use of a failure model that did not take execution times into consideration when scheduling tasks. Conversely, we propose a risk-free model integrating execution times with security levels and, therefore, our model can be leveraged to quantitatively measure quality of security. (6) We develop a practical security overhead model to estimate the computational overhead of commonly used security services like authentication and integrity.

In our previous work, we proposed an array of security-aware scheduling algorithms for a cluster [15] [17], a Grid [16] and storage systems [9]. Although these algorithms can improve system security and performance, the algorithms have no inherent capability of supporting heterogeneous clusters. In this regard, we are motivated to introduce the concept of security heterogeneity, and to propose a security and heterogeneity driven scheduling algorithm to improve security of real-time parallel applications running on het-

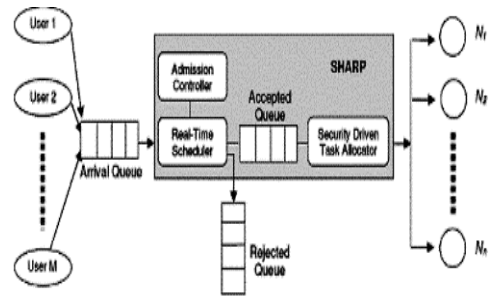


Figure 1: Security and Heterogeneity Driven Scheduling Architecture

erogeneous clusters.

The rest of the paper is organized in the following way. Section 2 shows preliminaries. Section 3 proposes the real-time scheduling algorithm for parallel applications with security and timing constraints. The objective function, risk-free probability, is derived in Section 5 presents performance results in comparison with existing algorithms. The paper concludes with Section 6.

2 Preliminaries

2.1 Security Driven Scheduling Architecture

In this section, we focus on a scheduling architecture for parallel applications with security and real-time constraints. As depicted in Figure 1, a heterogeneous cluster is comprised of n heterogeneous nodes connected via a high-performance network (e.g. Myrinet or fast Ethernet [13]) to process parallel applications submitted by users. Note that throughout this paper terms application and job are used interchangeably. Let $N = (N1, N2, \dots, Nn)$ denote the set of heterogeneous nodes. The scheduling architecture consists of a real-time scheduler, an admission controller, and a security driven task allocator. The admission controller is deployed to perform feasibility check by determining if an arriving parallel application can be completed by the heterogeneous cluster before the specified deadline. The parallel application will be admitted into the system if the application's deadline can be met by the cluster. The real-time scheduler is to satisfy timing requirements of parallel applications by assigning high priorities to applications with early deadlines. The security driven task allocator is intended to generate task allocation decision for each task of a parallel application, satisfying both security and real-time requirements.

2.2 Modeling Security Heterogeneity

We consider a class of embarrassing parallel applications (see [14] for some examples) each of which can be envisioned as a set of tasks without any interaction between one another. Our security driven scheduling approach can be easily extended to integrate a message scheduler, thereby being able to support communication-intensive parallel applications.

Suppose there is a parallel application submitted by a user, the application is modeled as a tuple (T, a, f, d, l) , where $T = (t_1, t_2, \dots, t_m)$ represents a set of 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 $E_i = (e_{i1}^1, e_{i2}^2, \dots, e_{in}^n)$ 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, $S_i = (s_i^1, s_i^2, \dots, s_i^q)$ a vector of security levels, characterizes the security requirements of the task. s_i^k ($1 \leq k \leq q$) is the normalized security level of the k^{th} security service required by t_i .

Let w_i^j denote the computational weight of task t_i on node N_j . is computed as a ratio between its execution time on N_j and that on the fastest node in the cluster. Thus, we have $w_i^j = e_i^j / \max_{k=1}^n (e_i^k)$. 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:

$$H_i^c = \sqrt{\frac{1}{n} \sum_{j=1}^n (w_i^{avg} - w_i^j)^2} \quad (1)$$

where $w_i^{avg} = (\sum_{j=1}^n w_i^j) / n$. The computational heterogeneity of a parallel application with task set T is calculated as:

$$H^c = \frac{1}{n} \sum_{T_i \in T} H_i^c \quad (2)$$

Besides computation heterogeneity, a cluster may exhibit security heterogeneity. While each task imposes requirements on a set of security services, each node in the cluster provides an array of security services with various quality of security, which is measured by security levels normalized in the range from 0.1 to 1.0. Security services provided by node N_j is

characterized as a vector of security levels, $P_j = (p_j^1, p_j^2, \dots, p_j^r)$, where p_j^k ($1 \leq k \leq r$) is the security level of the k^{th} security service provided by N_j .

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

$$H_i^s = \sqrt{\frac{1}{n} \sum_{j=1}^q (s_i^{avg} - s_i^j)^2} \quad (3)$$

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

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

$$H^s = \frac{1}{n} \sum_{t_i \in T} H_i^s \quad (4)$$

The heterogeneity of the k^{th} security service in a heterogeneous cluster is expressed as:

$$H_k^v = \sqrt{\frac{1}{n} \sum_{i=1}^n (p_{avg}^k - p_i^k)^2} \quad (5)$$

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

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

$$H_j^m = \sqrt{\frac{1}{n} \sum_{k=1}^m (p_j^{avg} - p_j^k)^2} \quad (6)$$

where $p_j^{avg} = (\sum_{k=1}^m p_j^k) / m$.

Using Equation (5), the heterogeneity in security services of the cluster can be computed as:

$$H^v = \frac{1}{n} \sum_{k=1}^q H_k^v \quad (7)$$

2.3 Heterogeneity in Security overhead

Now we consider heterogeneity in security overhead caused by security services. The following security overhead model account for three security services, including encryption, integrity, and authentication [15]. The security overhead model is general in that it can be easily extended to incorporate other security services.

Suppose task t_i requires q security services, which are provided in sequential order. Let s_i^k and $c_{ij}^k(s_i^k)$ be the security level and overhead of the k^{th} security service, the security overhead c_{ij} experienced by t_i

on node N_j can be computed using Equation (8). In particular, the security overhead of task t_i with security requirements for the three services is modeled by Equation (9).

$$c_{ij} = \sum_{k=1}^q c_{ij}^k(s_i^k) \quad (8)$$

where $s_i^k \in S_i$.

$$c_{ij} = \sum_{k \in (a,e,g)} c_{ij}^k(s_i^k) \quad (9)$$

where $s_i^k \in S_i$.

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, encryption, and integrity services [15]. Finally, the security overhead of a parallel application with task set T is calculated by:

$$c = \sum_{t_i \in T} \sum_{j=1}^n x_{ij} c_{ij} = \sum_{t_i \in T} \sum_{j=1}^n \left(x_{ij} \sum_{k \in (a,e,g)} c_{ij}^k(s_i^k) \right) \quad (10)$$

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

The authentication overhead can be obtained in Table 1.

The encryption overhead c_i^e of T_i on M_j is computed using Equation (11), where π_i^e is the CPU time spent in encrypting security sensitive data.

$$c_{ij}^e(s_i^e) = \pi_{ij}^e s_i^e \quad (11)$$

where $s_i^e \in S_i$

The integrity overhead can be calculated using the following equation, where l_i is the amount of security sensitive data, and $\mu^g(s_i^g)$ is a function mapping a security level into its corresponding integrity service performance.

$$c_{ij}^g(s_i^g) = l_i / \mu^g(s_i^g) \quad (12)$$

where $s_i^g \in S_i$

Table 1: Authentication Overhead. (s_i^a : Security Level, $c_i^a(s_i^a)$: Computation Time (ms))

Authentication Methods	s_i^a	$c_i^a(s_i^a)$
HMAC-MD5	0.3	90
HMAC-SHA-1	0.6	148
CBC-MAC-AES	0.9	163

2.4 Modeling Quality of Security

We build a model to quantitatively measure quality of security provided by a heterogeneous cluster. To achieve this goal, we introduce a closed form expression for the security benefit of task t_i under a given allocation. The security benefit of t_i is measured by Security Deficiency (SD), which is quantified as the discrepancy between the requested security levels and the security levels offered. Suppose task t_i is allocated to node N_j , the SD value of the k^{th} requested security service is defined as the following expression:

$$g(s_i^k, p_j^k) = \begin{cases} 0 & \text{if } s_i^k \leq p_j^k \\ s_i^k - p_j^k & \text{Otherwise} \end{cases} \quad (13)$$

For the k^{th} security service, a small SD value indicates a high degree of service satisfaction. In particular, a zero SD value implies that t_i 's requirement placed on the k^{th} security service can be perfectly met. The SD value of task t_i on node N_j can be derived from Expression (14). 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 the following equation to calculate the SD value of t_i on N_j :

$$SD(s_i, P_j) = \sum_{k=1}^q [w_i^k g(s_i^k, P_j^k)] \quad (14)$$

where w_i^k is the weight of the k^{th} security service, and $0 \leq w_i^k \leq 1$. Note that users specify in their requests the weights to reflect relative priorities given to the required security services.

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:

$$\begin{aligned} DSD(T, X) &= \sum_{t_i \in T} \sum_{j=1}^n [x_{ij} SD(s_i, p_j)] \\ &= \sum_{t_i \in T} \sum_{j=1}^n x_{ij} \sum_{k=1}^q [w_i^k g(s_i^k, p_j^k)] \end{aligned} \quad (15)$$

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

3 The SHARP Algorithm

We propose in this section the SHARP scheduling algorithm, which incorporates security requirements into the process of scheduling for heterogeneous clusters. Before we proceed to present the security driven scheduling algorithm, we formulate the scheduling problem as follows.

3.1 Security Driven Scheduling Problem Formulation

Let X be the schedule for all the tasks in task set T , we have $\forall x_{ij} \in x_i, \sum_{j=1}^n x_{ij} = 1 : x_{ij} = 1$ if t_i is allocated to node N_j . The scheduling decision of the task set T is optimal if (1) all the tasks can be completed before the deadline d , and (2) the degree of security deficiency (see Equation 15) is minimized. Since a security driven scheduler makes an effort to reduce the degree of security deficiency of each submitted parallel application, the following function needs to be minimized:

Minimize

$$DSD(T, X) = \sum_{t_i \in T} \sum_{j=1}^n x_{ij} \sum_{k=1}^q [w_i^k g(s_i^k, p_j^k)] \quad (16)$$

Subject to $f_i \leq d$

where f_i is the finish time of the i^{th} task in the task set.

Given a heterogeneous cluster and a sequence of submitted parallel applications, the proposed SHARP scheduling 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:

Minimize

$$\sum_{\forall T} DSD(T, X) \quad (17)$$

Substituting Equation (16) into (17) yields the following security objective function. Thus, our SHARP algorithm strives to schedule applications in a way to minimize the average degree of security deficiency of all schedulable parallel applications:

$$DSD = \frac{\sum_{\forall T} \{P_{sd}(T) \sum_{t_i \in T} \sum_{j=1}^n [x_{ij} \sum_{k=1}^q [w_i^k g(s_i^k, p_j^k)]]\}}{\sum_{\forall T} P_{sd}(T)} \quad (18)$$

where $P_{sd}(T)$ is a step function, and

$$P_{sd}(T) = \begin{cases} 1 & \text{if task set } T \text{ can be timely completed} \\ 0 & \text{Otherwise} \end{cases}$$

3.2 Properties of the Algorithm

Now we list some properties of the SHARP algorithm, which will be used in the subsequent subsection to construct the security driven scheduling algorithm. The schedule of a parallel application is feasible if all

tasks in its task set can be completed before its deadline. Specifically, a parallel application has a feasible schedule on a heterogeneous cluster if there exists a set of nodes, where a valid schedule can be generated for each task. This fact can be express by the following property.

Property 1. If a parallel application with task set T and deadline d has a feasible schedule on a heterogeneous cluster with n sites denoted by a set N , the following inequality must be satisfied:

$$\forall T \in T : \exists N_j \in N : \sigma_i^j + e_i^j + \sum_{k=1}^q c_{ij}^k(\bar{s}_i^k) \leq d$$

and,

$$\bar{s}_i^k = \begin{cases} s_i^k & \text{if } s_i^k \leq P_j^k \\ P_j^k & \text{Otherwise} \end{cases}$$

where σ_i^j is the start time of the i^{th} task on node N_j , is the computation time of the i^{th} task on N_j and $\sum_{k=1}^q c_{ij}^k(\bar{s}_i^k)$ is the security overhead experienced by the task. Note that the security overhead of the k^{th} security service relies on the security level (\bar{s}_i^k), which equals to s_i^k if node N_j can ensure high quality of k^{th} security service to satisfy the task's security needs. (\bar{s}_i^k) becomes p_i^k when N_j fails to fulfil t_i 's security requirements. Property 1 formally describes timing constraints, e.g., all the tasks of a parallel application must be completed before its deadline.

The earliest start time σ_i^j is one of the key factors in Property 1, and σ_i^j can be computed as :

$$\sigma_i^j = \tau + \sum_{t_l \in m, d_l \leq d} \left(e_l^j + \sum_{k=1}^q c_l^k(\bar{s}_i^k) \right) \quad (19)$$

where τ is the current time, and $\sigma_i^j = \tau + \sum_{t_l \in m, d_l \leq d} \left(e_l^j + \sum_{k=1}^q c_l^k(\bar{s}_i^k) \right)$ 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 N_j , the start time σ_i^j is the earliest available time of t_i on N_j .

3.3 Algorithm Description

The SHARP algorithm is outlined in Figure 2. 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 Equation 15) of each admitted parallel application is minimized.

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

```

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  $N_j$  in the heterogeneous cluster do
5.     Use Equation 19 to compute  $\sigma_j^i$ , the earliest start time of  $t_i$  on node  $N_j$ ;
6.     Calculate  $t_i$ 's security overhead  $\sum_{k=1}^q c_{ij}^k(\bar{s}_i^k)$ , where  $\bar{s}_i^k = \begin{cases} s_i^k, & \text{if } s_i^k \leq p_j^k \\ p_j^k, & \text{otherwise} \end{cases}$ ;
7.     if  $\sigma_j^i + e_j^i + \sum_{k=1}^q c_{ij}^k(\bar{s}_i^k) \leq d$  then (See Property 1)
8.       Use Equation 14 to compute the security deficiency  $SD(s_i, P_j)$  of task  $t_i$  on node  $N_j$ ;
9.       if  $SD(s_i, P_j) < SD_i$  then
10.         $SD_i \leftarrow SD(s_i, P_j)$ ;
11.         $x_j \leftarrow 1$ ;  $\forall k \neq j: x_k \leftarrow 0$ ;
12.      end if
13.    end for
14.  end for
15.  if no feasible schedule is available for  $t_i$  then
16.    Reject the parallel application, since:
17.  end for
18.  if all the tasks of the parallel application can be finished before deadline  $d$  then
19.    for each task  $t_i$  of the parallel application do
20.      allocate task  $t_i$  to node  $N_j$ , subject to  $|E_j^i| \leq n, x_j = 1$ ;
21.    end for
22.  end if

```

Figure 2: The SHARP scheduling algorithm

can be accomplished by calculating the earliest start time (see Equation 19) and the security overhead of each task (see Equations 8 and 9) 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 there is no way of guaranteeing the deadline of a task in the parallel application, 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. Now we derive the time complexity of the SHARP scheduling algorithm as shown in figure 2.

Theorem 1. The time complexity of SHARP is $O(nmq)$, where n is the number of nodes in a cluster, m 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 n nodes and m tasks, Steps 5-13 are executed for nm times. Therefore, the time complexity of Steps 2-17

Table 2: Characteristics of System Parameters

Parameter	Value (Fixed) - (Varied)
Number of tasks	(6400) - The first three month trace data from SDSC SP2 log
Number of nodes	(64)
Laxity	(4)-(1,2,3,4,5,6,7,8,9,10,15,20,30,40,50)1000seconds
Job arrival rate	Decided by the trace
Site security level (uniform dist.)	(0.1 - 1.0)
Application security	level (uniform dist.) (0.1 - 1.0) heightRequired security services Encryption, Integrity and Authentication
Weights security	services Authentication weight=0.2, Encryption weight=0.5, Integrity weight=0.3

is bounded by $O(nmq)$. Steps 18-22 take $O(m)$ time to allocate m task to n nodes in the cluster. Thus, the time complexity of SHARP is $O(1+nmq+m) = O(nmq)$.

4 Performance Results and Comparison

In purpose of revealing the strength of SHARP, we compared it with two well-known real-time scheduling 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*: A scheduling algorithm that schedules a ready job with the earliest deadline. If any algorithm is able to schedule a task set, *EDF* will be able to schedule it.
2. *LLF*: A scheduler that assigns priority based on laxity of jobs. Job with minimum laxity is assigned highest priority. Laxity = Deadline - Worst case computation time.

Section 5.1 describes performance metrics that we used and important parameters that will be examined in this section. Section 5.2 is to examine the overall performance improvements of SHARP over the two heuristics. The heterogeneous cluster simulator was designed and implemented based on the model and the algorithm described in the previous sections. Ta-

ble 2 summarizes the key configuration parameters of the simulated cluster used in our experiments.

4.1 Simulation Parameters

Before presenting the empirical results in detail, we present the simulation model as follows. 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. To simplify our experiments, we utilized the first three months data with 6400 parallel tasks in our 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* (see Equation 25), *Average degree of security deficiency* (see Equation 18), *Guarantee Ratio* (measured as a fraction of total submitted parallel applications that are found to be schedulable).

Since both security heterogeneity and computational heterogeneity are essential characteristics of security-critical heterogeneous clusters, we need to differentiate security overhead from traditional execution time. Specifically, we refer to execution time without security overhead as computational time, and the sum of computational time and security overhead total execution time.

4.2 Overall Performance Comparisons

The focus of this set of experiments is to compare the SHARP algorithm against the two alternative algorithms, and to understand the performance impacts of laxity and slack time on SHARP. The computational heterogeneity and the security heterogeneity are 1.08, and 0.22, respectively.

Figure 3 shows simulation results for the three algorithms on a cluster with 64 nodes. We observe that SHARP significantly outperforms the two competitors in terms of risk-free probability, degree of security deficiency, and guarantee ratio. For example, SHARP increases the risk-free probability by an average of 164.25% (see Figure 3a), whereas SHARP achieves improvement in degree of security deficiency by an average of 357.2% (see Figure 3b). We attribute the security improvement of SHARP over *EDF* and *LLF* to the fact that SHARP is a security driven scheduler and judiciously assigns tasks of a parallel applica-

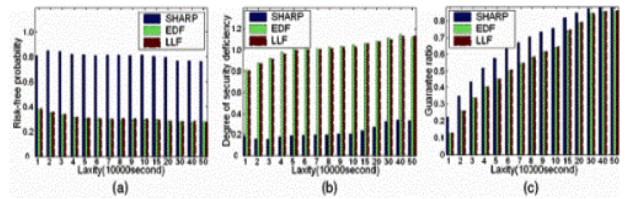


Figure 3: Performance impact of deadline

tion to a group of nodes that can minimize execution times while increasing quality of security. With regard to guarantee ratio, SHARP is noticeably better than *EDF* and *LLF* for all tested cases. For example, the average improvement in guarantee rate is 15.8% (see Figure 3c). This is because for each arrived application SHARP selects a set of nodes that lead to short total execution times while meeting timeliness constraints. On the contrary, *EDF* and *LLF* solely assign applications to a group of nodes offering the shortest computational time, thereby being unable to optimize total execution times. Consequently, compared with the SHARP algorithm, *EDF* and *LLF* generate fewer feasible schedules for submitted parallel applications.

When laxity goes up, the values of degree of security deficiency increase (see Figure 3b). This is because SHARP can admit more applications when deadlines become loose. Under a high workload condition, however, there exist fewer candidate nodes that can fulfil security demands while timely completing accepted applications. We notice in Figure 3a that the risk-free probability of SHARP slightly decreases with the increasing value of laxity. This result can be explained by two facts: (1) the risk-free probability is derived from both degree of security deficiency and total execution time (see Equation 25), and (2) a high degree of security deficiency results in a low value of risk-free probability.

One important implication derived from this set of experiments is that SHARP is not only a security driven heuristic, but also a heterogeneity driven algorithm offering high performance for parallel jobs on heterogeneous clusters.

5 Summary and Future Work

6. Summary and Future Work In this paper, we presented a security-driven scheduling algorithm for real-time parallel applications running on heterogeneous clusters. The main contributions of this study include: (1) an analysis of parallel applications with security and timing constraints; (2) a security

overhead model for parallel applications on heterogeneous clusters; (3) two new performance metrics (degree of security deficiency and risk-free probability) used to quantitatively measure quality of security provided by heterogeneous clusters; (4) considerations of heterogeneity in security and computation; (5) a security-and heterogeneity-driven scheduling algorithm (SHARP for short); (6) a simulation tool for large-scale heterogeneous clusters where the SHARP algorithm is implemented and evaluated.

The simulation results presented in the previous section demonstratively show that the degree of security deficiency can be used to increase the risk-free probabilities of feasible schedules produced by the SHARP scheduling algorithm. Our experimental results confirm that the SHARP algorithm can be employed to improve both security and performance of real-time parallel applications running on heterogeneous clusters. In future research, the heuristic will be extended to schedule data-intensive applications. This work will be accomplished by factoring in data transmissions among computational and disk I/O nodes. Further research in security-driven scheduling will be needed to address the issue of securing a variety of computing resources.

Acknowledgments 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.

References

- [1] A. Amin, R. Ammar, and A. El Dessouly, "Scheduling real time parallel structures on cluster computing with possible processor failures," *Proc. 9th Int'l Symp. Computers and Communications*, pp. 62 - 67, June 2004.
- [2] 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.
- [3] M. Bishop, *Computer Security*, ISBN 0-201-44099-7, Addison-Wesley, 2003.
- [4] 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.
- [5] A. Dogan and F. Zgner, "LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. Int'l Conf. Parallel Processing*, pp.352-359, B.C., Canada, 2002.
- [6] G. Donoho, "Building a Web Service to Provide Real-Time Stock Quotes," *MCAD.Net*, February, 2004.
- [7] M. Maheswaran and H.J. Siegel, "A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. the Seventh Heterogeneous Computing Workshop*, pp.57-69, 1998.
- [8] F. Petrini and W.-C. Feng, "Scheduling with Global Information in Distributed Systems," *Proc. 20th Int'l Conf. Distributed Computing Systems*, pp. 225 - 232, April 2000.
- [9] Mais Nijim, Xiao Qin, Tao Xie, Mohammed Alghamdi, "Awards: An Adaptive Write Scheme for Secure Local Disk Systems," *The 25th IEEE Int'l Performance Computing and Communications Conf.*, April 10-12, 2006, Phoenix, Arizona, USA.
- [10] 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.
- [11] X. Qin and H. Jiang, "Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems," *Proc. 30th Int'l Conf. Parallel Processing*, pp.113-122, Sept. 2001.
- [12] 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.
- [13] 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, Sept. 2004.
- [14] Wilkinson, B. and Allen M., "Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers", *Prentice-Hall, Inc.*, 1999.
- [15] T. Xie, X. Qin, and Andrew Sung, "SAREC: A Security-Aware Scheduling Strategy for Real-Time Applications on Clusters," *Proc. 34th Int'l Conf. Parallel Processing*, Norway, June 2005.
- [16] T. Xie and X. Qin, "Enhancing Security of Real-Time Applications on Grids through Dynamic Scheduling," *Proc. 11th Workshop Job Scheduling Strategies for Parallel Processing*, MA, June 2005.
- [17] 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*, September 27-30, 2005, Boston, USA
- [18] W. Yurcik, X. Meng, G. Koenig, J. Greenesid "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.