J. Parallel Distrib. Comput. 67 (2007) 1067– 1081

Performance Evaluation of a New Scheduling Algorithm for Distributed Systems with Security Heterogeneity

Tao Xie[a] Xiao Qin[b]

[a]*Department of Computer Science, San Diego State University, San Diego, CA 92182, USA*
[b]*Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA*

Abstract

High quality of security is increasingly critical for applications running on heterogeneous distributed systems. However, existing scheduling algorithms for heterogeneous distributed systems disregard security requirements of applications. To address this issue, in this paper, we introduce security heterogeneity concept for our scheduling model in the context of distributed systems. Based on the concept, we propose a novel heuristic scheduling algorithm, or SATS, which strives to maximize the probability that all tasks are executed without any risk of being attacked. Extensive experimental studies using real world traces indicate that the scheduling performance is affected by heterogeneities of security and computational power. Additionally, empirical results demonstrate that with respect to security and performance, the proposed scheduling algorithm outperforms existing approaches under a wide spectrum of workload conditions.

1. Introduction

A heterogenous distributed system consists of a collection of loosely coupled diverse computers, called sites, which are geographically distributed and connected by a communications network. Prime examples of distributed systems include web services [9][20] and peer-to-peer systems [7][12]. Over the last decade, heterogenous distributed systems have been emerging as popular computing platforms for computationally intensive applications with diverse computing needs [7]. To date they have been applied to security sensitive applications, such as banking systems and digital government, which require new approaches to security. Inherently, distributed systems are more vulnerable to threats than centralized systems, since it is difficult to control processing activities of the distributed systems and information can be accessed over networks. A variety of techniques like authentication [11] and access control [19] are widely used to secure distributed systems. Although these techniques can be applied to distributed systems, the conventional security techniques lack the ability to express heterogeneity in security services. Our study is intended to introduce a concept of security heterogeneity, which provides a means of measuring overhead incurred by security services in the context of heterogeneous distributed systems.

Scheduling algorithms play a key role in obtaining high performance in parallel and distributed systems [15][26]. Scheduling algorithms have been extensively studied in the past. Scheduling algorithms fall into two camps: static [7][10][16][21][22] and dynamic [2][17][21][27][28]. The objective of scheduling algorithms is to map tasks onto sites and order their execution in a way to optimize overall performance. In this work we consider the issue of dynamic task scheduling.

Nowadays, a wide variety of scheduling algorithms for distributed systems have been reported in the literature [1][4]. Peng and Shin proposed a new scheduling algorithm for tasks with precedence constraints in distributed systems [14]. Arpaci-Dusseau introduced two key mechanisms in implicit coscheduling for distributed systems [1]. Lo studied the problem of static task assignment in distributed systems [13]. The above algorithms were designed for homogeneous distributed systems.

In recent years, the issue of scheduling on heterogeneous distributed systems has been addressed and reported in the literature [6][25]. Ranaweera and Agrawal developed a scalable scheduling scheme called STDP for heterogeneous systems [18]. Srinivasan and Jha incorporated reliability cost, defined to be the product of processor failure rate and task execution time, into scheduling algorithms for tasks with precedence constraints [24]. A. Dogan and F. Özgüner studied reliable matching and scheduling for tasks with precedence constraints in heterogeneous distributed systems. Due to the lack of security awareness, these algorithms are not suitable for security-sensitive distributed computing applications.

Very recently, Song et al. proposed security-driven scheduling algorithms for grids [23]. This study is by far the closest to the proposed algorithm found in the literature. The main difference between our study and theirs are four-fold. (1) 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. (2) Our algorithm takes into account of heterogeneities in security and computation while theirs only considered homogeneous computing resources. (3) 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. (4) 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 a family of dynamic security-aware scheduling algorithms for a single machine [28], a cluster [26] and a Grid [27]. We conducted simulations to show that the proposed algorithms can consistently improve overall system performance in terms of quality of security and system schedulability. Unfortunately, these scheduling algorithms only support homogeneous computing applications, thus limiting their applicability to heterogeneous distributed systems. Hence, we are motivated in this study to formalize the security heterogeneity concept, and to propose a scheduling algorithm to improve security of heterogeneous distributed systems while minimizing computational overhead.

2. Modeling Tasks and Their Security Requirements

2.1 System model

In this study, we consider a queuing architecture of an n-site distributed system in which n heterogeneous sites are connected via a network to process independent tasks submitted by m users. Let $M = \{M_1, M_2, \ldots, M_n\}$ denote the set of heterogeneous sites. The system model, depicted in Figure 1, is composed of a task schedule queue, *STAS* task scheduler, and n local task queues. The function of *STAS* is intended to make a good task allocation decision for each arrival task to satisfy its security requirements and maintain an ideal performance in conventional performance metrics such as average response time.

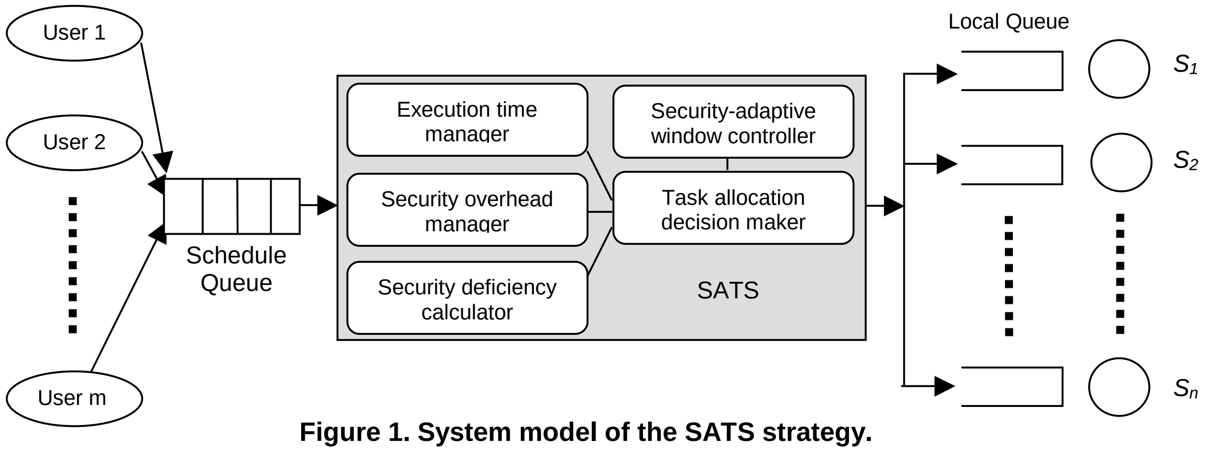

User 1
User 2
User m
Schedule Queue
Execution time manager
Security overhead manager
Security deficiency calculator
Security-adaptive window controller
Task allocation decision maker
SATS
Local Queue
S_1
S_2
S_n

Figure 1. System model of the SATS strategy.

A schedule queue is used to accommodate incoming tasks. *SATS* scheduler then processes all arrival tasks in a *First-Come First-Served (FCFS)* manner. After being handled by SATS, the tasks are dispatched to one of the designated site $M_i \in M$ for execution. The sites, each of which maintains a local queue, can execute tasks in parallel. The main component of the system model above is SATS, which is composed of five modules: (1) Execution time manager; (2) Security overhead manager; (3) Degree of security deficiency (*DSD*) calculator; (4) Security-adaptive window controller; and (5) Task allocation decision maker. Since execution time of each task can be estimated by code profiling and statistical prediction [3], we assume that the execution time of each arrival task for each site is a prior and this information is managed in the execution time manager module. Similarly, we assume that the security overhead for each arrival task on each site is a prior, and this information is maintained in the security overhead manager module. The *DSD* calculator is used to calculate discrepancies between an arrival task's security level requirements and the security levels that each site offers. The function of security-adaptive window controller is to vary size of the window to discover a suitable site for the current arrived task so that (1) its security demands can be well met; (2) the total execution time can be as small as possible. To illustrate how security-adaptive window controller works, we give an example as below.

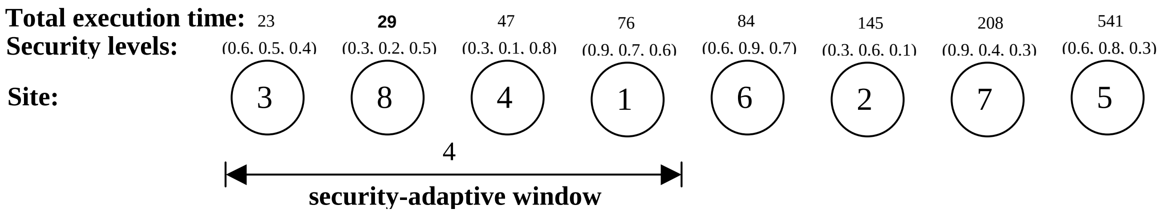


Figure 2. Example sorted site list with security-adaptive window 4.

In Figure 2 we assume that there are 8 sites in the system. The first row shows the total execution time for an arrival task on the 8 sites in seconds. The second row displays the security levels that the 8 sites can offer for three security services, namely, authentication, encryption, and integrity. The third row is a site list sorted by the task's total execution time in a non-decreasing order. The size of security-adaptive window is 4, which means SATS will select a site that can deliver the best security within the first four candidate sites. If security-adaptive window controller cannot find an idea site in terms of security, it will automatically enlarge the window to expand the search range. However, large window size will result in a long total execution time for the task in a high probability because the total execution time increases when the size of the window enlarges.

After retrieving information like execution time on each site, security overhead on each site, degree of security deficiency on each site and the size of security-adaptive window for the current task from the corresponding modules, the task allocation decision maker will decide which site will be assigned to the task.

Each site in the system model above is inherently heterogeneous in both computation and security. Computational heterogeneity means that for each task the execution time on different sites is distinctive. While each task has an array of security service requests, each site offers the security services with different levels. The level of a security service provided by a site is normalized in the range from 0.1 to 1.0. Suppose site M_j, offers q security services, $P_j = (p_j^1, p_j^2, \ldots, p_j^q)$, a vector of security levels, characterizes the security levels provided by the site. p_j^k is the security level of the kth security service provided by M_j. To meet security requirement, security overhead of the task will be considered. Security heterogeneity suggests that the security overhead of a task varies on each site.

2.2 Modeling tasks with security requirements

We consider a class of heterogeneous distributed systems where an application is comprised of a collection of tasks performed to accomplish an overall mission. It is assumed that tasks are independent of one another. Each task requires a set of security services with various security levels specified by a user. Values of security levels are normalized in the range from 0.1 to 1.0 as well. For example, a task specifies in its request security level 0.7 for the authentication service, 0.3 for the integrity service, and 0.8 for the encryption service. Note that the same security level value in different security services may have various meanings.

Suppose there is a task T_i submitted by a user, T_i is modeled as a set of rational parameters, e.g., $T_i = (a_i, E_i, f_i, l_i, S_i)$, where a_i and f_i are the arrival and finish times, and l_i denotes the amount of data (measured in MB) to be protected. E_i is a vector of execution times for task T_i on each site in M, and $E_i = (e_i^1, e_i^2, \ldots, e_i^n)$. Suppose T_i 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. s_i^k is the security level of the kth security service required by T_i.

A security-aware scheduler has to make use of a function to measure the security benefits gained by each arrival task. In particular, the security benefit of task T_i is quantitatively modeled as a function of the discrepancy between security levels requested and the security levels offered. The security benefit function for task T_i on site M_j is denoted by $DSD: (S_i, P_j) \rightarrow \Re$, where $\Re$ is the set of non-negative real numbers:

$$DSD(s_i) = \sum_{k=1}^{q} w_i^k g(s_i^k, p_j^k)\text{, where } 0 \le w_i^k \le 1,\ \sum_{k=1}^{q} w_i^k = 1\text{, and } g(s_i^k, p_j^k) = \begin{cases} 0, \text{if } s_i^k \le p_j^k \\ s_i^k - p_j^k, \text{otherwise} \end{cases} \quad (1)$$

Note that w_i^k is the weight of the kth security service for task T_i. Users specify in their requests the weights to reflect relative priorities given to the required security services. *Degree of Security Deficiency*, or *DSD,* is defined to be a weighted sum of q discrepancy values between security levels requested by a task and the security levels offered by a site. For each task, a small *DSD* value means a high satisfaction degree. Zero *DSD* value implies that a task's security requirements can be perfectly met. That is, there exists at least one site M_j in M that can satisfy the following condition:

$$\forall k \in [1, q], s_i^k \le p_j^j$$

Let X_i be all possible schedule for task T_i, $x_i \in X_i$ be a scheduling decision of T_i. Given a task T_i, the *degree of security deficiency value* (DSDV) of T_i is expected to be minimized:

$$DSDV(x_i) = \min_{x_i \in X_i}\{DSD(x_i)\} = \min_{x_i \in X_i}\left\{\sum_{k=1}^{q} w_i^k (g(s_i^k, p^k(x_i)))\right\}, \tag{2}$$

where $p^k(x_i) = p_j^k$ if the task is allocated to site *j*. A security-aware scheduler strives to minimize the system's overall DSDV value defined as the sum of the degree of security deficiency of submitted tasks (See Equation 1). Thus, the following *DSDV* function needs to be minimized:

$$SDSD(x) = \min_{x \in X}\left\{\sum_{T_i \in T} DSDV(x_i)\right\}, \tag{3}$$

where T is a set of submitted tasks. Substituting Equation (2) into (3) yields the following security objective function. Thus, our proposed SATS scheduling algorithm makes an effort to schedule tasks in a way to minimize Equation (4): .

$$SDSD(x) = \min_{x \in X}\left\{\sum_{T_i \in T}\left(\min_{x_i \in X_i}\left\{\sum_{k=1}^{q} w_i^k (f(s_i^k, p^k(x_i)))\right\}\right)\right\} \tag{4}$$

Since the degree of security deficiency for task T_i merely reflects the security service satisfaction degree experienced by the task, it is inadequate to measure quality of security for T_i during its execution. Therefore, we derive in this section the probability $P_{rf}(T_i, M_j)$ that T_i remains risk-free during the course of its execution.

The quality of security of a task T_i with respect to the *k*th security service is calculated as $\exp\left(-\lambda_i^k\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l(s_i^l)\right)\right)$ where λ_i^k is the task's risk rate of the *k*th security service, and $c_{ij}^l(s_i^l)$ is the security overhead experienced by the task on site *j*. The risk rate is expressed as:

$$\lambda_i^k = 1 - \exp(-\alpha(1 - s_i^k)). \tag{5}$$

Note that this model assumes that risk rate is a function of security levels, and the distribution of risk-free for any fixed time interval is approximated using a *Poisson* probability distribution. The risk rate model is just for illustration purpose only. Thus, the model can be replaced by any risk rate model with a reasonable parameter α.

The quality of security of task T_i on site M_j can be obtained below by considering all security services provided to the task. Consequently, we have:

$$P_{rf}\left(T_i, M_j\right) = \prod_{k=1}^{q} \exp(-\lambda_i^k \left(e_i^j + \sum_{l=1}^{q} c_{ij}^l (s_i^l) \right)) = \exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l (s_i^l) \right) \sum_{k=1}^{q} \lambda_i^k \right). \tag{6}$$

Using equation (6), we obtain the overall quality of security of task T_i in the system as follows,

$$P_{rf}\left(T_i\right) = \sum_{j=1}^{n} \left\{ P\left[x_i = j\right] \cdot P_{rf}\left(T_i, M_j\right) \right\} = \sum_{j=1}^{n} \left\{ p_{ij} \cdot \exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l (s_i^l) \right) \sum_{k=1}^{q} \lambda_i^k \right) \right\}, \tag{7}$$

where p_{ij} is the probability that T_i is allocated to site M_j. Given a task set T, the probability that all tasks are free from being attacked during their executions is computed based on Equation (7). Thus,

$$P_{rf}(T) = \prod_{T_i \in T} P_{rf}\left(T_i\right). \tag{8}$$

By substituting the risk rate model into Equation (8), we finally obtain $P_{rf}(T)$ as shown below:

$$P_{rf}(X) = \prod_{T_i \in T} \left\{ \sum_{j=1}^{n} \left\{ p_{ij} \cdot \exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l (s_i^l) \right) \sum_{k=1}^{q} \lambda_i^k \right) \right\} \right\}. \tag{9}$$

In summary, DSD values show us security service satisfaction degrees experienced by tasks, while risk-free probability measured by Equation (9) defines quality of security provided by a heterogenous distributed system. In Section 5 these two metrics are used to evaluate security of distributed systems.

2.3 Heterogeneity model

The computational weight of task T_i on site M_j (e.g., w_i^j) is defined as a ratio between its execution time on M_j and that on the fastest site in the system. That is, we have $w_i^j = e_i^j \Big/ \min_{k=1}^{n}\left(e_i^k\right)$. The computational heterogeneity level of task T_i, referred to as H_i^C, can be quantitatively measured by the standard deviation of the computational weights. Formally, H_i^C is expressed as:

$$H_i^C = \sqrt{\frac{1}{n} \sum_{j=1}^{n} \left(w_i^{avg} - w_i^j \right)^2}, \text{ where } w_i^{avg} = \left(\sum_{j=1}^{n} w_i^j \right) \Big/ n. \tag{10}$$

The computational heterogeneity of a task set T can be computed by $H^C = \frac{1}{|T|} \sum_{T_i \in T} H_i^C$.

There are three types of security heterogeneities. (i) Security heterogeneity of a particular task T_i indicates the difference of security requirements in the q security services requested by the task (see Equation 11). (ii) Security heterogeneity of a given security service provided by each site in a distributed system reflects the discrepancy of the offered security levels of the service in the system (see Equation 12). (iii) Security heterogeneity of a particular site M_j shows the deviation of the q

security levels provided by the site (see Equation 13).

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 measured by the standard deviation of the security levels in the vector. Thus,

$$H_i^S = \sqrt{\frac{1}{q}\sum_{j=1}^{q}\left(s_i^{avg} - s_i^j\right)^2}\text{, where } s_i^{avg} = \left(\sum_{j=1}^{q} s_i^j\right)\Big/ n. \quad (11)$$

The security requirement heterogeneity of a task set T can be computed by $H^S = \frac{1}{|T|}\sum_{T_i \in T} H_i^S$.

The heterogeneity level of the kth security service in a distributed system is defined as:

$$H_k^V = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(p_{avg}^k - p_i^k\right)^2}\text{, where } p_{avg}^k = \left(\sum_{i=1}^{n} p_i^k\right)\Big/ n. \quad (12)$$

Using Equation (12), the heterogeneity of security services can be written as $H^V = \frac{1}{q}\sum_{k=1}^{q} H_k^V$.

Finally, the heterogeneity level of security services in site M_j is defined to be:

$$H_j^M = \sqrt{\frac{1}{q}\sum_{k=1}^{q}\left(p_j^{avg} - p_j^k\right)^2}\text{, where } p_j^{avg} = \left(\sum_{k=1}^{q} p_j^k\right)\Big/ q. \quad (13)$$

2.4 Security overhead model

Now we consider security overhead incurred by security services. The following security overhead model includes three services, namely, encryption, integrity, and authentication [26]. The security overhead model can be easily extended to incorporated more 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 kth security service, the security overhead c_{ij} experienced by T_i on site M_j can be computed using Equation (14). In particular, the security overhead of T_i with security requirements for the three services above is modelled by Equation (15).

$$c_{ij} = \sum_{k=1}^{q} c_{ij}^k(s_i^k)\text{, where } s_i^j \in S_i. \quad (14)$$

$$c_{ij} = \sum_{k \in \{a, e, g\}} c_{ij}^k(s_i^k)\text{, where } s_i^j \in S_i. \quad (15)$$

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 [26]. The authentication overhead can be obtained using the following table.

Table 1. Authentication Overhead

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

Our security level assignment is reasonable because a security mechanism providing higher quality of security imposes higher overhead than mechanisms offering lower security. Please note that the security level assignments in Table 1 (s_i^a =0.3, 0.6, 0.9) are only for illustration purpose. The security level of a security mechanism can be quantitatively measured by the amount of cost needed to successfully break the mechanism. However, quantitatively measuring the security level of a security mechanism is a nontrivial research issue, and it is out of the scope of this work.

The encryption overhead c_i^e of T_i on M_j is computed using Equation (16), where π_i^e is the CPU time spent in encrypting security sensitive data.

$$c_{ij}^e(s_i^e) = \pi_{ij}^e s_i^e \text{, where } s_i^e \in S_i \tag{16}$$

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.

$$c_{ij}^g(s_i^g) = l_i / \mu^g(s_i^g) \text{, where } s_i^g \in S_i. \tag{17}$$

3. The SATS Algorithm

In Section 3 we proposed the SATS algorithm, which integrates security requirements into scheduling for heterogeneous distributed systems. For task T_i, the earliest start time on site M_j is $\text{es}_j(T_i)$, which can be computed by Equation (18).

$$\text{es}_j(T_i) = r_j + \sum_{T_l \in W_j} \left(e_l^j + \sum_{k=1}^{q} c_{lj}^k(s_l^k) \right), \tag{18}$$

where r_j represents the remaining overall execution time of a task currently running on the *j*th site, and $e_l^j + \sum_{k=1}^{q} c_{lj}^k(s_l^k)$ is the overall execution time (security overhead is factored in) of waiting task T_l assigned to site M_j prior to the arrival of T_i. Thus, the earliest start time of T_i is a sum of the remaining overall execution time of the running task and the overall execution times of the tasks with earlier arrival on site M_j. Therefore, the earliest completion time for task T_i on site M_j can be calculated as:

$$\text{e}c_j(T_i) = es_j(T_i) + e_i^j + \sum_{k=1}^{q} c_{ij}^k(s_i^k) = r_j + \sum_{T_l \in W_j} \left(e_l^j + \sum_{k=1}^{q} c_{lj}^k(s_l^k) \right) + e_i^j + \sum_{k=1}^{q} c_{ij}^k(s_i^k) \tag{19}$$

The SATS algorithm is outlined in Figure 3. The goal of the algorithm is to deliver optimal quality of security while maintaining high performance for tasks running on heterogeneous systems. To achieve the goal, SATS manages to minimize degree of security deficiency (see Equation 1) of each task (see Step 10 in Fig. 3) without performance deterioration.

```
1.for each task $T_i$ submitted to the schedule queue do
2.     for each site $M_j$ in the system do
3.          Use Equation (18) to compute $es_j(T_i)$, the earliest start time of $T_i$ on site $M_j$;
4.          Use Equation (19) to compute $ec_j(T_i)$, the earliest completion time of $T_i$ on site $M_j$
5.     end for
6.    Sort all sites in earliest completion time in a non-decreaseing order
7.    for each site in the security-adaptive window do
8.          Use Equation (1) to compute $DSD(s_i)$ /* Compute degree of security deficiency for each site*/
9.    end for
10.   Select the site $M_r$ that can offer the smallest DSD value and assign $T_i$ on it
11.   Update site $M_r$'s earliest available time $es_j$
12.   Use Equation (6) to compute  risk-free probability for task $T_i$
13.   Record start time and completion time for task $T_i$
14.end for
```

Figure 3. The SATS algorithm.

Before optimizing the degree of security deficiency of task T_i, SATS sorts all the sites in a non-decreasing order in T_i's total execution time based on the information retrieved from the execution time manager and the security overhead manager (see Figure 1). Step 7 computes the degree of security deficiencies for the task on each site in the light of the security deficiency calculator. Combining the input from the security-adaptive window controller, the task allocation decision maker decides a site to which the task is allocated.

We have the following theorem for the time complexity of the proposed SATS algorithm.

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

Proof. The time complexity of computing the earliest start time and the earliest completion time for task T_i on a site is *O(m)* (Steps 3 and 4). Sorting the earliest completion time a non-decreasing order (Step 6) will take *O(mlgm)* since we only have *m* sites. To compute the risk-free probability, the time complexity is *O(3k)* (Step 12). For other steps, they only consume *O(1)*. Thus, the time complexity of the SATS algorithm is express as follows: O(*n*)(*O(m)* + *O(mlgm)*+ *O(3k))* = *O(nmlgm)*.

4. Simulations

In Section 3 we proposed the SATS strategy, which integrates security requirements into scheduling for dynamic tasks running on distributed systems. Now we are in a position to evaluate the effectiveness of SATS by comparing its performance with two well-known scheduling heuristics in both security-related and conventional metrics.

Using extensive simulation experiments based on San Diego Supercomputer Center (SDSC) SP2 log, we evaluate in this section the potential benefits of the SATS algorithm. The real trace was sampled on a 128-node (66MHz) IBM SP2 from May 1998 through April 2000. To simplify our experiments, we utilized the first three months data with 6400 parallel tasks in simulation. Since the trace was sampled from a homogenous environment, to reflect the heterogeneity of the simulated distributed system, we translated the "execution time" of each task from a single value to a vector with *n* (number of sites) elements based on the heterogeneity model described in Section 2.3. In purpose of revealing the strength of SATS, we compared it with two well-known scheduling algorithms, namely, *Min-Min* and *Sufferage* [23]. *Min-Min* and *Sufferage* are non-preemptive task scheduling algorithms, which schedule a stream of independent tasks onto a heterogeneous distributed computing system. They are representative dynamic scheduling algorithms for distributed systems and were successfully applied in real world distributed resources management systems such as SmartNet. The two algorithms are briefly described below.

(1) *MINMIN*: For each submitted task, the site that offers the earliest completion time is tagged. Among all the mapped tasks, the one that has the minimal earliest completion time is chosen and then allocate to the tagged site.

(2) *SUFFERAGE*: Allocating a site to a submitted task that would "suffer" most in terms of completion time if that site is not allocated to it.

The purpose of comparing SATS with MINMIN and SUFFERAGE is to show the performance improvements over existing task scheduling algorithms in a distributed computing environment. Section 4.1 describes performance metrics that we used and important parameters that will be examined in this section. Section 4.2 is to examine the performance improvements of SATS over the two heuristics. We will investigate in Section 4.3 the performance impacts of heterogeneity of both computation and security in a simulated 16-site distributed system. Section 4.4 studies the performance sensitivity of the SATS algorithm to the size of data to be secured. We evaluate the scalability of the proposed SATS algorithm in terms of the size of a distributed system.

A competitive advantage of conducting simulation experiments is that performance evaluation on a distributed system can be accomplished without additional hardware cost. The distributed system simulator was designed and implemented based on the model and the algorithm described in the previous sections. Table 2 summarizes the key configuration parameters of the simulated distributed system used in our experiments.

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 sites	(16) – (8, 16, 32,64,128)
Task arrival rate	Decided by the trace
Size of security-adaptive window	(8) – (1, 2, 4, 8, 16)
Data to be secured (uniform dist.)	(1–100) – (0.01–1,1 – 100, 10 – 1000, 100 – 10000, 1000 –100000) MB
Site security level (uniform dist.)	(0.1 – 1.0)
Task security level (uniform dist.)	(0.1 – 1.0)
Required security services	Encryption, Integrity and Authentication
Weights security services	Authentication weight=0.2, Encryption weight=0.5, Integrity weight=0.3
Computational heterogeneity	(1.08) – (0, 0.43, 1.08, 1.68, 2.27)
Security heterogeneity	(0.22) – (0, 0.14, 0.22, 0.34, 0.56)

4.1 Simulator and Simulation Parameters

Before presenting the empirical results in detail, we present the simulation model as follows. The parameters of sites in the simulated distributed system are chosen to resemble real-world workstations like IBM SP2 nodes.

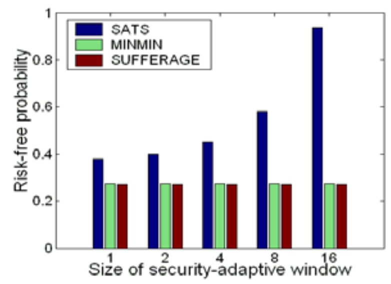
We modified the trace by adding a block of data to be secured for each task in the trace. The size of the security-required data assigned to each task is controlled by a uniform distribution (see Table 2). Although “task number”, “submit time”, and “execution time” of tasks submitted to the system are taken directly from the trace, “size of data to be secured”, “number of sites”, “computational heterogeneity”, and “security heterogeneity” are synthetically generated in accordance with the above model since they are not available in the trace. The performance metrics we used include: *risk-free probability* (see Equation 9), *degree of security deficiency* (see Equation 1), *site utilization* (defined as the percentage of total task running time out of total available time of a given site), *makespan* (the latest task completion time in the task set), *average response time* (the response time of a task is the time period between the task’s arrival and its completion and the average response time is the average value of all tasks’ response time), *slowdown ratio* (the slowdown of a task is the ratio of the task’s response time to its service time and the slowdown ratio is the average value of all tasks’ slowdowns).

4.2 Overall Performance Comparisons

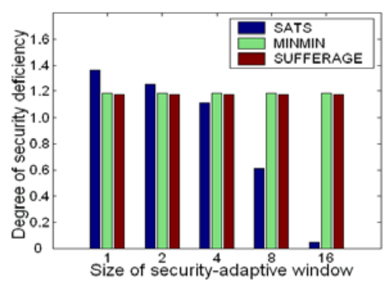
The goal of this experiment is two fold: (1) to compare the proposed SATS algorithm against the two heuristics, and (2) to understand the sensitivity of SATS to the size of security-adaptive window.

Note that tasks arrived in the system require the three security services.

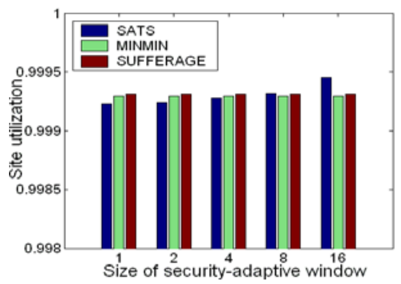
Figure 4 shows the simulation results for the three algorithms on a distributed system with 16 sites. Since MINMIN and SUFFERAGE do not have a security-adaptive window, their performance in all six metrics keeps constant. We observe from Figure 4 (a) that SATS significantly outperforms the two heuristics in terms of risk-free probability, whereas MINMIN and SUFFERAGE algorithms exhibit similar performance. We attribute the performance improvement of SATS over MINMIN and SUFFERAGE to the fact that SATS is a security-adaptive scheduler and judiciously assigns a task to a site not only considering its computational time but also its security demands. When the size of security-adaptive window (*saw*) increases, the value of risk-free probability goes up. This is because SATS can select a more appropriate site in terms of meeting the task's security demands when *saw*, which is the number of candidate sites to be chosen, becomes larger.



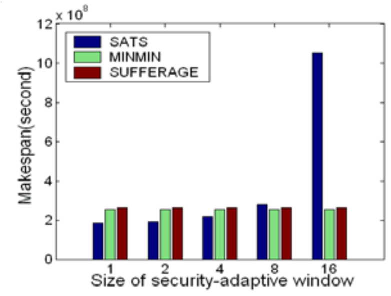
(a) Risk-free probability



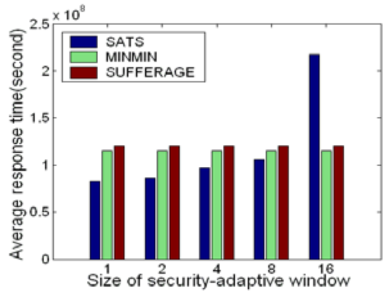
(b) Degree of security deficiency



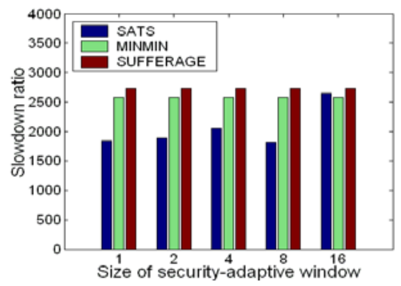
(c) Site utilization



(d) Makespan in seconds



(e) Average response time in seconds



(f) Slowdown ratio

Figure 4. Performance impact of size of security-adaptive window.

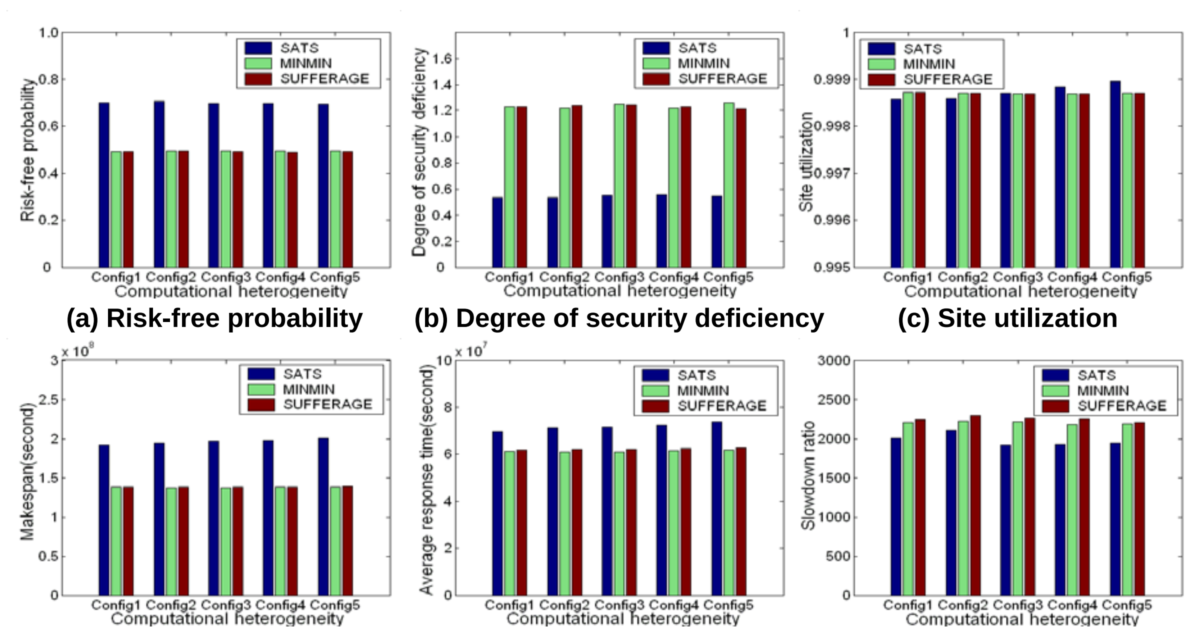
We notice in Figure 4 (b) that SATS performs poorly in terms of *degree of security deficiency* (DSD) when *saw* is small, while it still outperforms the two baseline algorithms with respect to *risk-free probability*. This interesting result can be explained by the fact that the value of *risk-free probability* is derived from both DSD and total execution time including security overhead (see Equation 9). Although the values of DSD are greater than the two heuristics, SATS results in a much shorter total execution time when *saw* is slim. This is because a candidate site in the small *saw* provides a task with a shorter total execution time. When *saw* becomes large, SATS significantly outperforms the two alternatives in DSD for SATS can dispatch tasks to sites that guarantee the tasks' security requirements. From site utilization standpoint, Figure 4 (c) shows that SATS will eventually outperforms the two heuristics when *saw* increases. This is because when SATS tries to allocate a task in case where *saw* is large, SATS gives higher priority to security requirements. Consequently, SATS assigns the tasks to sites that meet the security constraints at the cost of long total execution times.

Figures 4d-4f plot the performance results of the three algorithms in terms of classical performance metrics, including makespan, average response time, and slowdown ratio. As we can see from the figures, SATS is superior to the other heuristics when the *saw* is not in its maximal size. This happens because SATS only selects sites with shorter total execution time when *saw* is small. When the value of *saw* is 16, however, SATS exhibits long makespan and response times (see Figure 4d and Figure 4e). The rationale behind this is that SATS compares all of the 16 sites to identify one that can best meet an arrival task's security demands regardless of the total execution time on this site. On the contrary, when *saw* is small, SATS simply assign an arrival task to a site that is able to fulfil the task's security needs. In addition, SATS chooses a site providing a relatively short total execution time.

One important implication derived from this experiment is that SATS is a security-aware and adaptive task scheduler, which dynamically adjusts its *saw* to consistently make good balancing between security and performance.

4.3 Impact of heterogeneity of computation and security

As we mentioned in Section 2, in a heterogeneous distributed system, the computational times of a particular task on different sites are distinctive, which is referred to as computational heterogeneity. Besides, the security overheads incurred by the task on different sites are diverse. We name this security heterogeneity. We believe that both computational and security heterogeneities are essential characteristics of security-sensitive heterogeneous distributed systems. To differentiate execution time from security overhead, we use the term computational time to refer execution time without security overhead, and the total execution time of a task consists of both computational time and security overhead. In this experiment we investigate the impact of these two heterogeneities on system performance. Specifically, we evaluate the performance of the three algorithms in cases where (1) security heterogeneity keeps constant while computational heterogeneity varies (see Figure 5); (2) computational heterogeneity is fixed but security heterogeneity is increased (see Figure 6); and (3) both heterogeneities varies (see Figure 7).



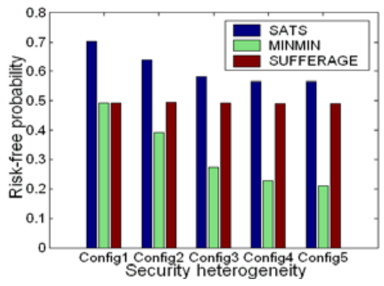
(a) Risk-free probability **(b) Degree of security deficiency** **(c) Site utilization**

(d) Makespan in seconds **(e) Average response time in seconds** **(f) Slowdown ratio**

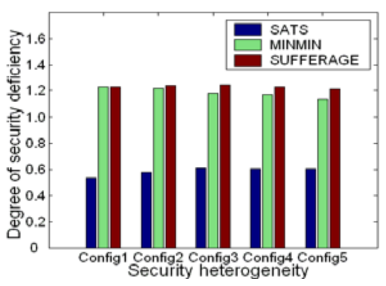
Figure 5 Performance impact of computational heterogeneity.

In Figure 5, Config1-Config5 represents 5 different computational heterogeneity values listed in Table 2. Figure 5b reveals that SATS consistently delivers better performance in DSD than the other two heuristics. This is because SATS is a security-adaptive scheduler, meaning that it makes an effort to adjust the value of *saw* that yields the optimized quality of security. Unlike SATS, MINMIN and SUFFERAGE are unaware of tasks' security needs, thereby allocating tasks to sites based on computational times. Therefore, there is a strong likelihood to assign tasks to sites that lead to a large value of DSD. The result shown in Figure 5a can be explained by the result in Figure 5b because a small value of DSD results in a high *risk-free probability* provided that total execution time is a constant. Figures 5d and 5e show the downside of SATS. Since within the *saw* range SATS picks a site that can best serve a task's security demands and, therefore, SATS is likely to select a site where the total execution time is not minimal. Thus, the *makespan* and the *average response time* of SATS are longer than the other two heuristics. This can be explained by the following two reasons. First, MINMIN and SUFFERAGE tend to select a site with a small computational time without considering a task's security overhead. Second, the security overhead of the task remains the same on all sites. Hence, small computational times imply small total execution times. Figure 5 suggests that SATS can achieve a high performance in two security-related metrics while performing poorly in conventional metrics. This result demonstrates that SATS is not suitable for distributed systems with low level of security heterogeneity.

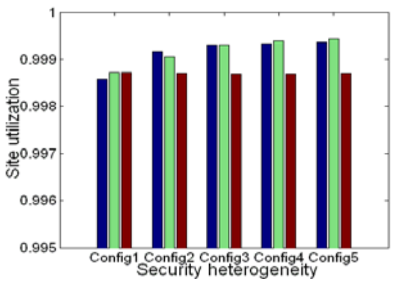
Now we evaluate impact of security heterogeneity in Figure 6, which shows that SATS is superior to the other two heuristics in both the security-related metrics in all cases and the conventional metrics in most cases when security overhead varies among different sites. The reason is that MINMIN and SUFFERAGE do not factor in the security overhead of a task and therefore possibly assign the task to a site that only result in a small computational time but with a large total execution time, while SATS considers computational time, security overhead, and the security demands of the task when scheduling it, and thus, SATS can achieve a good performance in both metric camps. The implication of this experiment tells us that SATS can show its strength in a distributed system where its security heterogeneity varies.



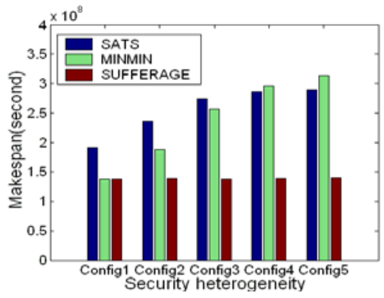
(a) Risk-free probability



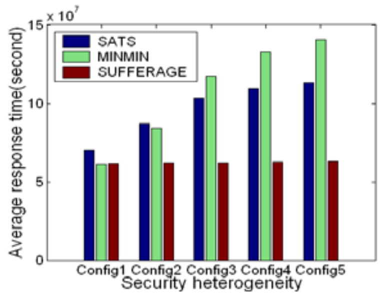
(b) Degree of security deficiency



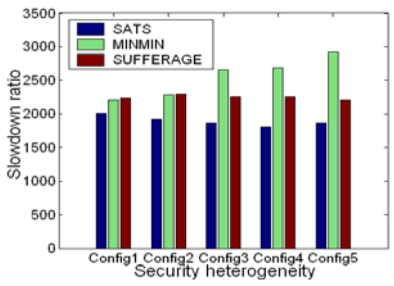
(c) Site utilization



(d) Makespan in seconds



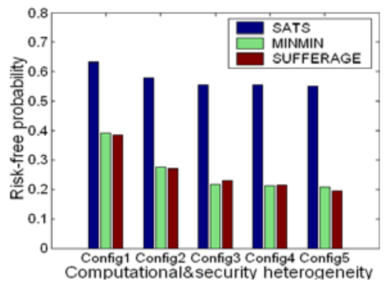
(e) Average response time in seconds



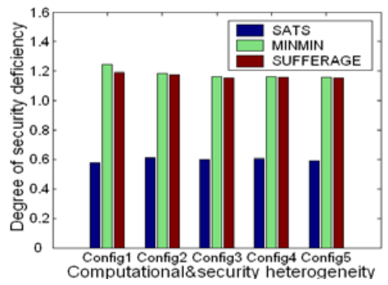
(f) Slowdown ratio

Figure 6. Performance impact of security heterogeneity.

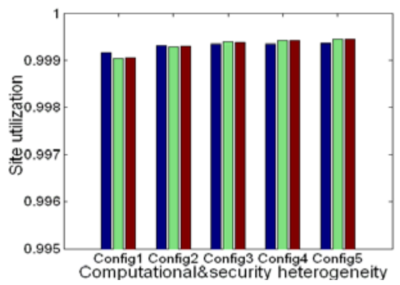
When both computational heterogeneity and security heterogeneity vary at the same time, SATS fully exhibits its power as shown in Figure 7. The average performance improvement in *risk-free probability* compared with MINMIN and S UFFERAGE is 120.8%. In terms of *degree of security deficiency*, on average SATS outperforms MINMIN and SUFFERAGE over 96.66% and 94.98, respectively. Still, SATS has noticeable performance improvement in terms of *makespan*, *average response time*, and *slowdown ratio*. The implication derived from Figure 7 convinces us that SATS is more appropriate than the other two heuristics in a security-critical distributed system where both computational time and security overhead are heterogeneous in nature.



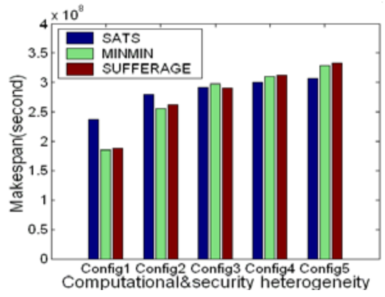
(a) Risk-free probability



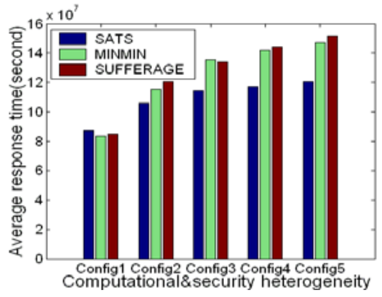
(b) Degree of security deficiency



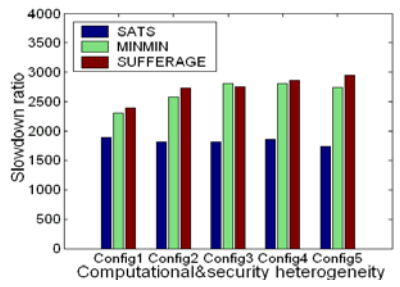
(c) Site utilization



(d) Makespan in seconds



(e) Average response time in seconds



(f) Slowdown ratio

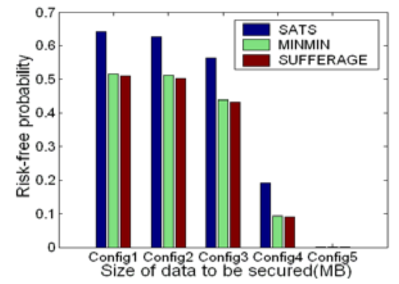
Figure 7. Performance impact of computational and security heterogeneity.

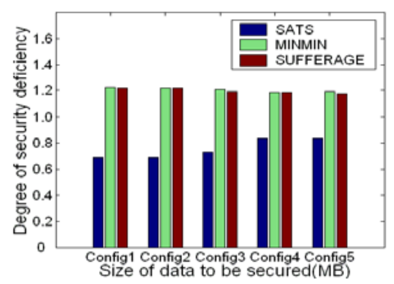
In summary, the strength of SATS can be fully demonstrated when a distributed system is heterogenous in both computation and security. When the system is only heterogeneous in computational time, SATS cannot deliver a comparable performance in conventional metrics but achieves a much higher result in security-related metrics. This fact hints us that SATS can significantly improve performance in almost all six metrics without increasing hardware cost when a distributed system is heterogeneous in both computation and security, which is the case in reality. This is the beauty of SATS. The results suggest that SATS can be successfully applied in an existing security-critical heterogeneous distributed system.

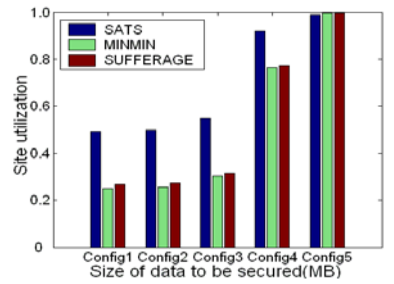
4.4 Sensitivities to size of data to be secured

In this experiment we evaluate the performance impact of size of data to be secured. As we described in previous sections, each task was synthetically assigned a block of data that needs to be protected from being disclosed or being tampered. The size of data to be secured will influence the task completion time because the larger the data, the longer time is needed to protect it. To discover the performance impact of data size, we tested five configurations (see Table 2).

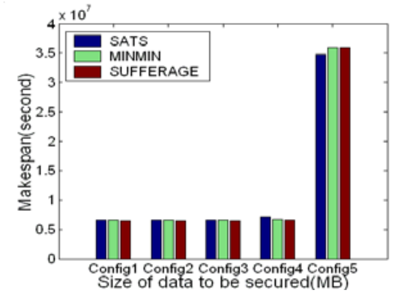
The experimental results are shown in Figure 8. There are several important observations based on Figure 8. Firstly, when the size of date to be secured increases, the *degree of security deficiency* of SATS also increases, while MINMIN and SUFFERAGE maintains the same (see Figure 8b). For SATS this is because sites with lower security levels are more likely to be chosen. The rationale behind is that security overhead becomes moderately dominant when the size of data to be secured exceeds a threshold. Therefore, the sites with short total execution time, which sit in the *saw* (remember that SATS sorts all sites in a non-decreasing order in terms of total execution times of a task before scheduling it), have smaller security overhead, which means security levels are lower in these sites. Thus, the discrepancy between the security levels offered by the selected site and the security levels demanded by the task enlarges.

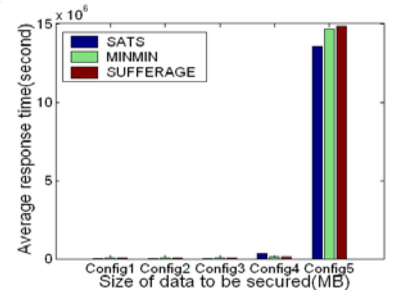


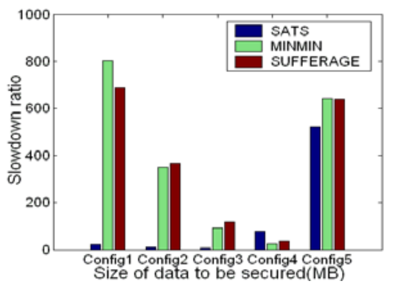




(a) Risk-free probability **(b) Degree of security deficiency** **(c) Site utilization**







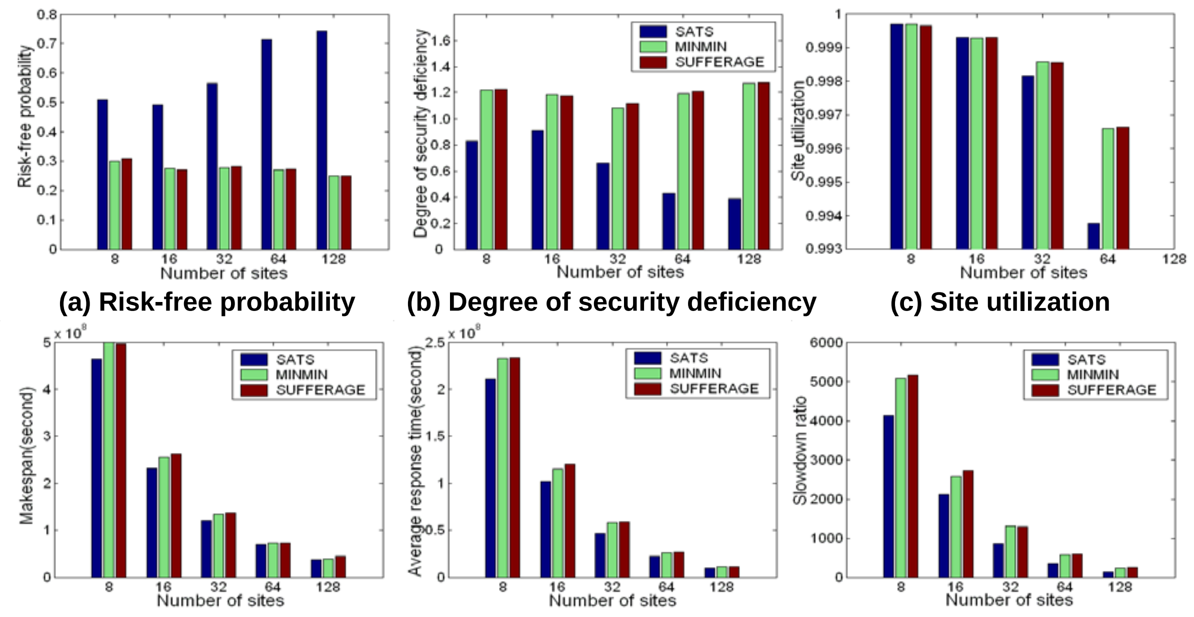
(d) Makespan in seconds **(e) Average response time in seconds** **(f) Slowdown ratio**

Figure 8. Performance impact of size of data to be secured.

As for MINMIN and SUFFERAGE, their values of *degree of security deficiency* keep constant. This is because they are not aware of the security overhead change and therefore still select a site with a small computation time. Unlike the candidate sites in the *saw* in *SATS* scenario, these selected sites still have a randomly distributed security levels. Hence, the values of DSD for MINMIN and SUFFERAGE do not change even when the size of data to be secured becomes large. Secondly, Figure 8a shows that SATS degrades its performance in *risk-free probability* when size of data to be secured enlarges. There are two reasons that are responsible for this result. One is that the value of DSD increases, and the other is that the total execution time increases. Thirdly, it is very interesting to see that there exits a big jump between Config4 and Config5 in terms of *makespan* and *average response time* for all three algorithms (see Figures 8d and 8e). This is because when the data size is in the range [1, 100] GB, a task's security overhead is comparable with its computation time. Thus, the total execution time substantially increases, resulting in a huge *makespan* and *average response time*.

4.5 Scalability

This experiment is intended to investigate the scalability of the SATS algorithm. We scale the number of sites in a heterogeneous distributed system from 8 to 128. Figure 9 plots the performances as functions of the number of sites in the simulated distributed system. The results show that the SATS approach exhibits good scalability.



(a) Risk-free probability (b) Degree of security deficiency (c) Site utilization

(d) Makespan in seconds (e) Average response time in seconds (f) Slowdown ratio

Figure 9. Performance impact of number of sites.

Figures 9a and 9b show the improvement of SATS in *risk-free probability* and DSD over the other two heuristics. It is observed from Figure 9a that the amount of improvement becomes more prominent with the increasing value of site number. This result can be explained by the non-security-awareness nature of MINMIN and SUFFERAGE, which merely select a site for a task without considering the task's security demands. Conversely, SATS can achieve a much higher performance when there are more sites available in the system. This is because with a high probability SATS can find a site that meets a task's security demands well when there are more sites to be chosen. For all the three algorithms, their performances improve in terms of *makespan*, *average response time*, and *slowdown ratio*. This can be readily understood because more sites result in a low value for *makespan*, *average response time*, and *slowdown ratio*.

5. Summary and Future Work

In this paper, we considered the security requirements of applications in the context of task scheduling in heterogeneous distributed systems. This is important because increasing number of applications running on heterogeneous distributed systems requires not only descent scheduling performance but also high quality of security. To solve this problem, we proposed a security-adaptive scheduling heuristic that is based on the concept of security heterogeneity. Experimental results demonstrate that our strategy outperforms existing approaches in both security and performance on a simulated heterogeneous distributed system.

In future research, the heuristic will be extended to schedule parallel applications. This work can be accomplished by factoring in precedence constraints among tasks and communication security. Further research will be needed to address the issue of securing other resources.

Acknowledgments

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 grant, Auburn University under a startup grant, the Intel Corporation under Grant No. 2005-04-070, and the Altera Corporation under an equipment grant.

References

[1] A. C. Arpaci-Dusseau, “Implicit Coscheduling: Coordinated Scheduling with Implicit Information in Distributed Systems,” *ACM Trans. on Computer Systems*, Vol.19, No. 3, pp.283-331, Aug. 2001.

[2] P. Barcaccia, M.A. Bonuccelli, and M. Di Ianni, “Complexity of Minimum Length Scheduling for Precedence Constrained messages in Distributed Systems,” *IEEE Trans. Parallel and Distributed Systems*, Vol.11, No.10, pp. 1090–1102, Oct. 2000.

[3] T. D. Braun et al., “A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems,” *Proc. Workshop Heterogeneous Computing*, pp.15-29, Apr. 1999.

[4] T.L. Casavant and J.G. Kuhl, “A Taxonomy of Scheduling in General-purpose Distributed Computing Systems,” *IEEE Trans. Software Engineering*, Vol.14, No.2, pp.141-154, Feb. 1988.

[5] H. Chen and M. Maheswaran, “Distributed dynamic scheduling of composite tasks on grid computing systems,” *Proc. Int’l Symp. Parallel and Distributed Processing*, pp.88-97, April 2002.

[6] A. Dogan and F. Özgüner, “Reliable matching and scheduling of precedence-constrained tasks in heterogeneous distributed computing,” *Proc. Int’l Conf. Parallel Processing*, pp. 307-314, 2000.

[7] A. Dogan and F. Özgüner, “LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems,” Proc. Int’l Conf. Parallel Processing, pp.352-359, B.C., Canada, 2002.

[8] R. Gupta and A. K. Somani, “An Incentive Driven Lookup Protocol for Chord-Based Peer-to-Peer (P2P) Networks,” *Proc. 11th Int’l Conf. High Performance Computing*, pp. 8-18, Bangalore, India, Dec. 2004,

[9] M. Harchol-Balter, N. Bansal, B. Schroder, and M. Agrawal, “SRPT Scheduling for Web Servers,” *Proc. 7th Int’l Workshop Job Scheduling Strategies for Parallel Processing*, pp.11-20, MA, June 2001.

[10] Y.K. Kwok and I. Ahmad, “Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors,” *ACM Computing Surveys,* Vol.31, No. 4, pp.406-471, 1999.

[11] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, “Authentication in distributed systems: Theory and practice,” *ACM Trans. Computer System*s, Vol.10, No.4, pp 265-310, Nov. 1992.

[12] X. Li and J. Wu, “A Hybrid Searching Scheme in Unstructured P2P Networks,” *Proc. the 34th Int’l Conf. Parallel Processing*, Norway, June 2005.

[13] V. M. Lo, “Heuristic Algorithms for Task Assignment in Distributed Systems,” *IEEE Trans. Computers*, Vol. 37, No.11, pp. 1384-1397, 1988.

[14] D.-T. Peng and K.G. Shin, “Optimal scheduling of cooperative tasks in a distributed system using an enumerative method,” *IEEE Trans. Software Engineering*, Vol.19, No.3, pp. 253-267, March 1993.

[15] 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.

[16] 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.

[17] 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.

[18] S. Ranaweera, and D.P. Agrawal, “Scheduling of Periodic Time Critical Applications for Pipelined Execution on Heterogeneous Systems,” *Proc. Int’l Conf. Parallel Processing*, pp. 131-138, Sept. 2001.

[19] R. S. Sandhu, et al. “Role-Based Access Control Models,” IEEE Computer, Vol.29,No.2, pp.38-47, 1996.

[20] M. D. Santo, N. Ranaldo, and E. Zimeo, “Kernel Implementations of Locality-Aware Dispatching Techniques for Web Server Clusters,” *Proc. IEEE Int’l Conf. Cluster Computing*, Dec. 2003.

[21] B. Shirazi, M. Wang, G. Pathak, “Analysis and Evaluation of Heuristic Methods for static task scheduling,” *Journal of Parallel and Distributed Computing*, Vol.10, No.3, pp.222-2232, Nov. 1990.

[22] J. M. P. Sinaga, H. H. Mohammed, and D. H. J. Epema, “A Dynamic Co-Allocation Service in Multicluster Systems,” *Proc. 10th Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.

[23] 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.

[24] S. Srinivasn and N. K. Jha, “Safty and Reliability Driven Task Allocation in Distributed Systems,” *IEEE Trans. Parallel and Distributed Systems*, Vol.10, No.3, pp. 238-251, Mar. 1999.

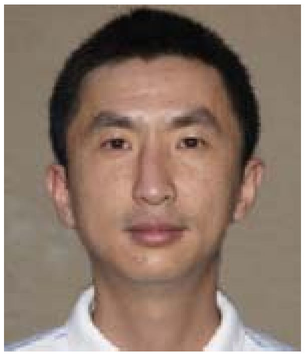
[25] H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and Low-complexity Task Scheduling

for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Sys.*, Vol.13, No.3, Mar. 2002

[26] 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.

[27] 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.

[28] T. Xie, X. Qin, A. Sung, M. Lin, and L. Yang, "Real-Time Scheduling with Quality of Security Constraints," *International Journal of High Performance Computing and Networking*, Feb. 2006.



Tao Xie received his Ph.D. degree in computer science from the New Mexico Institute of Mining
and Technology in 2006. He received his B.Sc. and M.Sc. 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, CA. His research interests are security-aware scheduling, high performance computing, cluster and Grid computing, parallel and distributed systems, real-time/embedded systems, and storage systems. He is a member of the IEEE.

Xiao Qin (S'99–M'04) received his BS and MS degrees in Computer Science from Huazhong University of Science and Technology, China, in 1996 and 1999, respectively. He received his Ph.D. in Computer Science from 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, fault tolerance, and performance evaluation. He had served as a subject area editor of *IEEE Distributed System Online* (2000–2001). He has been on the program committees of various international conferences, including IEEE Cluster, IEEE IPCCC, and ICPP.