A Novel Security-Driven Scheduling Algorithm for Precedence-Constrained Tasks in Heterogeneous Distributed Systems

Xiaoyong Tang, Kenli Li, Zeng Zeng, and Bharadwaj Veeravalli, *Senior Member*, *IEEE*

Abstract—In the recent past, security-sensitive applications, such as electronic transaction processing systems, stock quote update systems, which require high quality of security to guarantee authentication, integrity, and confidentiality of information, have adopted heterogeneous distributed system (HDS) as their platforms. This is primarily due to the fact that single parallel-architecture-based systems may not be sufficient to exploit the available parallelism with the running applications. Most security-aware applications end up in handling dependence tasks, also referred to as Directed Acyclic Graph (DAG), on these HDSs. Unfortunately, most existing algorithms for scheduling such DAGs in HDS fail to fully consider security requirements. In this paper, we systematically design a security-driven scheduling architecture that can dynamically measure the trust level of each node in the system by using differential equations. To do so, we introduce *task priority rank* to estimate security overhead of such security-critical tasks. Furthermore, we propose a security-driven scheduling algorithm for DAGs which can achieve high quality of security for applications. Our rigorous performance evaluation study results clearly demonstrate that our proposed algorithm outperforms the existing scheduling algorithms in terms of minimizing the makespan, risk probability, and speedup. We also observe that the improvement obtained by our algorithm increases as the security-sensitive data of applications increases.

Index Terms—Directed acyclic graphs, scheduling algorithm, security overheads, heterogeneous distributed systems, security-driven, precedence-constrained tasks.

1 Introduction

With the advent of new high-speed networks, it is now possible to connect a collection of distributed, cost-effective, and possibly heterogeneous resources in the form of a computational environment, such as cluster [1], Grid [2], multihop wireless network [3], etc. Heterogeneous distributed systems (HDSs) are usually composed of diverse sets of resources with different capabilities and normally interconnected with arbitrary networks to meet the requirements of widely varying applications. Over the last decade, HDSs have been emerging as popular computing platforms for compute-intensive applications with various computing needs [4]. In particular, numerous security-sensitive applications, such as electronic transaction processing systems, stock quote update systems, E-commerce online services, and digital government, have started to employ HDSs as their platforms [5], [6].

Unfortunately, since distributed systems are built to execute a broad spectrum of unverified user-implemented applications from different users, both of the applications and users can be sources of security threats to the system [7]. For example, the vulnerability of applications can be exploited by hackers to penetrate distributed systems very easily and malicious users can access these systems to launch denial-of-service attacks. Even a legitimate user may tamper with shared data or excessively consume system resources that may disrupt services performed by other HDSs users [7]. However, most existing distributed systems fail to consider any security mechanism to deal with these security threats [8]. Thus, it is necessary to deploy a set of security services to protect security-critical applications running on HDSs.

Although using security services can satisfy the applications' security needs, these services will lead to security overheads in terms of computational time, which will increase the applications' schedule length (makespan). The conflict between achieving good performance and high quality of security protection imposed by security-critical applications introduces a new challenge in resource allocation domain. Moreover, security heterogeneity and the dynamic property of HDSs makes solving this challenge more difficult, as the security overheads are node-dependent. That means different nodes can provide different security level for users, even at the same level of security service, and different amount of computation leads to distinct security overheads. However, the existing resource allocation schemes for parallel applications with precedence-constrained tasks in HDSs [9], [10], [11], [12], [13], [14] normally do not factor in applications' security requirements when making resource allocation decisions. That is far from enough for security-sensitive parallel applications. Motivated by this challenge, in this paper, we design and evaluate a security-driven scheduling model

- *X. Tang, K. Li, and Z. Zeng are with the School of Computer and Communication, Hunan University, Changsha 410082, China. E-mail: tang_313@163.com, lkl510@263.net, zeng_zeng@hotmail.com.*
- *B. Veeravalli is with the Computer Networks and Distributed Systems Laboratory, Department of Electrical and Computer Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 117576. E-mail: elebv@nus.edu.sg.*

Manuscript received 4 Nov. 2009; revised 3 Mar. 2010; accepted 28 Apr. 2010; published online 2 June 2010.
Recommended for acceptance by V. Barbosa.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-11-0561.
Digital Object Identifier no. 10.1109/TC.2010.117.

that can dynamically computes security overheads. Then, we propose a scheduling algorithm that integrates security services into that scheduling model for parallel applications with precedence-constrained tasks in HDSs.

Our contributions are multifold and can be summarized as follows: We propose, design and evaluate a security-driven scheduling architecture, which mainly includes, a Trust Manager, Trust Value Computation, Security Overhead Controller, Schedule Queue, Scheduler and Dispatch, running on HDSs. We create a trust method formulated using differential equations to dynamically compute the trust level of nodes in such HDSs. Further, we introduce security overhead model to quantitatively measure security overhead in terms of the computational time and analyze the system risk probability, an important metric of performance. Finally, we propose a security-driven scheduling algorithm for parallel applications with precedence-constrained tasks in such HDSs. This contribution in this paper is one of the first attempts to consider modeling the entire system with security implications and we attempt to carefully design a scheduler that subsumes security measures in executing a task. We believe that incorporating such security measures would aid designing more robust schedulers for high delay-sensitive and networked applications.

The rest of the paper is organized as following: In Section 2, we outline the related work in this domain. In Section 3, we describe the system architecture and the parallel application scheduling model. In Section 4, we propose a method that can dynamically compute the trust level of security service provided to each node, security overheads, and risk probabilities. In Section 5, we present a security-driven allocation scheme and investigates the properties in HDSs. We verify the performance of the proposed algorithm by comparing the results obtained from performance evaluation in Section 6 and conclude the paper in Section 7.

2 Related Work

Scheduling algorithms play a key role in obtaining high performance in HDSs. The objective of scheduling is to map tasks onto machines and order their executions so that task precedence requirements are satisfied with a minimal schedule length (makespan). A popular representation of a parallel application is the directed acyclic graph (DAG) in which the nodes represent application tasks and the directed arcs or edges represent intertask dependencies, such as task's precedence. It is widely known that the problem of finding the optimal schedule is NP-complete [15]. The most common heuristic DAG scheduling is the traditional list scheduling algorithm. However, most list scheduling algorithms are designed for homogeneous systems [16], [17], [18], [19], [20], [21], [22], [23]. Several list scheduling algorithms have been proposed to deal with heterogeneous systems, for example, mapping heuristic (MH) [9], dynamic-level scheduling (DLS) algorithms [10], levelized-min time (LMT) algorithm [11], Dynamic Critical Path (DCP) [12], Critical-Path-on-a-Machine (CPOP) algorithms and heterogeneous earliest-finish-time (HEFT) algorithm [13], [14]. HEFT algorithm significantly outperforms DLS algorithm, MH, LMT, and CPOP algorithms in terms of average schedule length ratio, speedup, etc. [13].

Most well-known scheduling approaches ignore security issues, and only few groups of researchers investigate the security-driven scheduling domain from different angles in various contexts. Dogan and Özgüner developed an efficient static scheduling algorithm (QSMTS_IP) for a heterogeneous computing system [24]. The QSMTS_IP algorithm is capable of meeting diverse QoS requirements including security for multiple users simultaneously, while minimizing the number of users whose tasks are failed to be scheduled due to resource limitations. In [25], [26], [27], [28], [29], Xie and Qin studied a family of dynamic security-aware scheduling algorithms for single machine, homogeneous cluster, and heterogeneous distributed system. Their studies addressed the applications' demands for both real-time performance and security. They conducted simulation experiments to show that the proposed algorithms can consistently improve overall system performance in terms of quality of security service and system schedulability. Unfortunately, these scheduling algorithms only support real-time applications and do not sufficiency evaluate node's trust. In recent years, dynamic trust mechanism is a new and hot topic of security research for HDS. Many trust models have been proposed, such as Sun's model [32], Eigen Trust model [36], Dimitri's model [33], PTM [46], Hassan's model [47], George's model [48], etc. Actually, these trust models are derived from the corresponding environments and not adequately consider all of the possible cases in HDS. On the other hand, few of them are realized in system. Azzedin and Maheswaran [30], [31] suggested to integrate the trust concept into Grid resource management. They proposed a trust model that incorporates the security implications into scheduling algorithms. However, their trust model and scheduling algorithm are not suitable for HDS.

A closest work to this research reported in a literature was accomplished recently by Song et al. [6], [34]. They developed three risk-resilient strategies and a genetic-algorithm-based scheme, that is, the Space Time Genetic Algorithm (STGA), to provide security assurance in grid job scheduling. However, their algorithms cannot be applied in HDSs for parallel applications with security requirements for two reasons: first, their methods to compute security overheads fails to support node's trust manager, and thus it cannot effectively evaluate security overhead in HDSs; second, their algorithms only consider batch scheduling, where jobs are independent of each other, and hence it cannot schedule parallel applications, where precedence constraints and communications among tasks exist in a single application. All of these studies are based on parallel applications with independent tasks. Actually, most HDS applications are intertask-dependent. Hence, we are motivated to incorporate security into intertask dependency scheduling, and to propose a scheduling algorithm to improve security of HDSs while minimizing the computational overheads.

3 System Architecture: Model and Definitions

In this section, we first introduce the architecture of the security-driven scheduling and then we outline the model of parallel application with precedence-constrained tasks.

TABLE 1
Definitions of Notations

Notation	Definition
ID	user identification
T	the set of v weighted tasks in the application
t_i	the ith task in the application
$w(t_i)$	the computational cost of task t_i
E	the set of weighted and directed edges representing communications among tasks in T
$e_{i,j}$	the directed edge from ith task to jth task
$w(e_{i,j})$	the communication cost of edge $e_{i,j}$
P	the set of m heterogeneous processors
p_i	the ith processor in the system
u_i	the execution speed of p_i
$SD^k_{i,j}$	the kth security requirement of task t_i for node p_j
$TL^k_{j,id}$	the kth trust level the node p_j provides for user id
$C^k_{i,j}$	the kth security overheads of task t_i for node p_j
$Pr(t^k_i, p_j)$	the kth risk probability of task t_i for node p_j
$Pr(t_i, p_j)$	the risk probability of task t_i for node p_j
$Pr(t_i)$	the risk probability of task t_i
$SRank(t_i)$	the security upward rank of task t_i
$succ(t_i)$	the set of immediate successors of task t_i
$pred(t_i)$	the set of immediate predecessor tasks of task t_i
$EST(t_i, p_j)$	the earliest computation start time of task t_i on processor node p_j
$EFT(t_i, p_j)$	the earliest computation finish time of task t_i on processor node p_j

Finally, we present our HDSs model that is used in this paper. For ease of understanding, we summarize the notations and their meanings used throughout of this paper in Table 1.

3.1 Security-Driven Scheduling Architecture

The general scheduling architecture is proposed in literature [25], [26], [34]. However, they are not effectively incorporate the security issues (trust model, security overhead, and security demand) into scheduling. Thus, we propose a security-driven scheduling architecture wherein the main difference from general architecture is that it includes the security evaluation modules such as Trust Manager, Trust Value Computation, and Security Overhead Controller, as depicted in Fig. 1. Trust Manager module is used to discover and collect security threats and performance metrics of every service provider (computational node) in HDSs. Some of the performance metrics are success rate, speedup, memory, bandwidth, reliability, and so on. Security threats mainly include: individual malicious peers, malicious collectives, malicious collectives with camouflage, malicious spies, sybil attack, man in the middle attack, driving down the reputation of a reliable peer, partially malicious collectives, malicious pretrusted peers [35]. Trust Value Computation is used to compute trust level based on the information from Trust Manager module, which uses some trust models, such as Bayesian [33], EigenTrust [36], and our new approach. Security Overhead Controller evaluates the security overheads for a task assigned to a node. Scheduler and Dispatch module is deployed to generate resource allocation decisions for each task in a parallel application to improve security of HDSs and to minimize the computational overheads, and then dispatches the task to the destination node. The Schedule Queue maintained by the admission controller is deployed to accommodate incoming applications. If workload is extremely high, the Schedule Queue become bottleneck of the system. This problem can be resolved by multiqueue techniques.

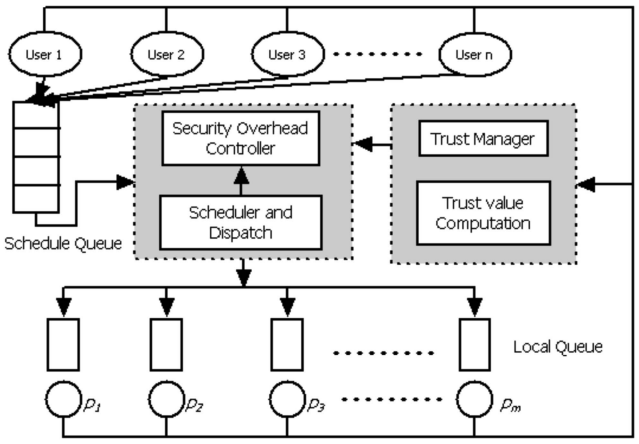


Fig. 1. Security-driven scheduling architecture.

3.2 Parallel Application Model

Generally, a parallel application model with precedence-constrained tasks is represented by a directed acyclic graph $G = <ID, T, E, SD>$, where ID is the identification of user; T is the set of v tasks that can be executed on any of the available nodes (processors); $E \in T \times T$ is the set of directed arcs or edges between the tasks to represent dependency. For example, edge $e_{i,j} \in E$ represents the precedence constraint that task t_i should complete its execution before task t_j starts the execution. A task may have one or more inputs. When all of the inputs are available, the task is triggered to execute. When the task is finished, it generates the required outputs. The weight $w(t_i)$ assigned to a task t_i represents the computational cost and the weight $w(e_{i,j})$ assigned to edge $e_{i,j}$ represents the communication cost. A task with no parent tasks in DAG is called an *entry task* and a task with no child task in DAG is called an *exit task*. In this paper, we assume that DAG has exactly one entry task t_{entry} and one exit task t_{exit}. If multiple *exit* tasks or *entry* tasks exist, they maybe connected with zero time-weight edges to construct a single pseudoexit task or a single pseudoentry task. Fig. 2a shows an example DAG with assigned task and edge weights.

SD is security demand of the parallel task to all available nodes. The security demand of task t_i for node p_j can be specified as a q-tuple $SD_{i,j} = (SD^1_{i,j}, SD^2_{i,j}, \ldots, SD^q_{i,j})$, where $SD^k_{i,j}$ represents the required security-level range of the

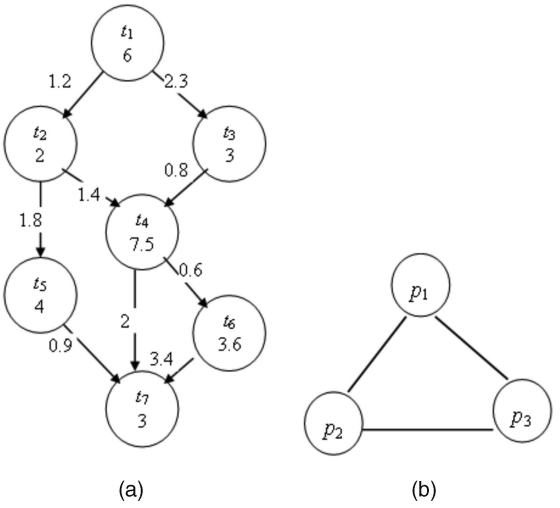


Fig. 2. An example of parallel application and heterogeneous distributed system.

kth security service ($1 \leq k \leq q$). Since snooping, alteration, and spoofing are three common attacks in HDSs [37], [38], we consider three security services, which are authentication service, integrity service, and confidentiality service, to protect HDSs. For example, snooping, which is an unauthorized interception of information, can be countered by confidentiality services, which encrypt data by using cryptographic algorithms and hence, hackers cannot correctly interpret the data. These three security services can be denoted as $SD^a_{i,j}$, $SD^g_{i,j}$, $SD^d_{i,j}$, respectively. SD is specified by users from very low to very high to represent their security requirements. For the convenience of calculation, these security requirements are normalized into the range $[0, 1]$. Such as, the very low security demand is 0.1, the mean is 0.5, and the very high is 1.0. Table 2 lists the security demands of seven tasks for three nodes as an illustrative example.

3.3 HDSs Model and Assumptions

In this paper, we consider heterogeneous computing environment model as a set P of m heterogeneous processors (nodes) connected via a fully connected network. A simple example of such a HDS is shown in Fig. 2b, in which the parameters of each machine are shown in Table 3. We will discuss these parameters and their values, in detail, in Section 4.2 after we introduce the trust model and the function that computes security overheads. Note that the trust level is dynamically computed by the system with a fixed frequency. We assume that:

- Any node can execute/compute the task and communicate with other machines at the same time. The communication speed between two machines is also set as 1 without loss of generality.
- Once a node has started task execution, it continues without interruptions, and after completing the execution it immediately sends the output data to all the children tasks in parallel.

TABLE 2
Example of Task Security Demands

Task	$SD^a_{i,1}$	$SD^g_{i,1}$	$SD^d_{i,1}$	$SD^a_{i,2}$	$SD^g_{i,2}$	$SD^d_{i,2}$	$SD^a_{i,3}$	$SD^g_{i,3}$	$SD^d_{i,3}$
t_1	0.4	0.1	0.6	0.2	0.3	0.1	0.2	0.5	0.5
t_2	0.2	0.3	0.2	0.6	0.1	0.1	0.5	0.3	0.4
t_3	0.8	0.4	0.7	0.3	0.2	0.5	0.1	0.2	0.6
t_4	0.1	0.1	0.3	0.4	0.4	0.4	0.3	0.7	0.5
t_5	0.4	0.2	0.4	0.1	0.7	0.3	0.4	0.4	0.4
t_6	0.2	0.7	0.5	0.8	0.3	0.2	0.3	0.3	0.3
t_7	0.6	0.5	0.1	0.9	0.2	0.9	0.4	0.5	0.5

TABLE 3
The Computational Node Parameters

Node	u_j	$TL^a_{j,1}$	$TL^g_{j,1}$	$TL^d_{j,1}$	λ^a_j	λ^g_j	λ^d_j
p_1	2	0.5	0.4	0.8	0.4	1.1	3
p_2	1	0.7	0.2	0.9	5.1	0.2	0.7
p_3	4	0.3	0.7	0.3	1.1	0.2	1.6

4 Security and Trust Requirements

As pointed in [30], [31], [33], [34], [35], [36], [37], [43], security and trust are two different notions. Security is a notion associated with the assurance of security computing services provided by system nodes. Whereas trust is reflected by the behavior of system nodes. These two terms are correlated by many attributes as discussed in [34]. In this section, first, we focus on how to evaluate the node's trustworthiness, named as the *trust level* (TL) of a node, using differential equations which are partly derived from our previous preliminary work in [43]. Then, we consider security demand (SD) of user tasks and analyze overall system security.

4.1 Definition of Trust and Reputation

The notion of trust is a complex subject relating to a firm belief in attributes such as reliability, honesty, firewall capabilities, antivirus capabilities, and the competence of the trusted entity. In this paper, we will consider the following trust definitions [30], [31], [32], [33], [34], [35], [36], [43]:

Definition 1. *Trust is a firm belief in the competence of a node to act as expected. In addition, this belief is not a fixed value associated with the node but rather it is subject to the user's behavior and can be applied only within a specific context at a given time.*

In other words, the *belief* is a dynamic value and spans over a set of values ranging from *very trustworthy to very untrustworthy*. The *trust level* is built on past experiences and given for a specific context. For example, a node p might trust user id to read its storage resources but it may not allow to write on these resources. The *trust level* is specified within a given time because the *trust level* now between two entities is not necessarily the same *trust level* as some time ago.

When making trust-based decisions, any node can rely on the information of a specific user. This is named as *reputation* that can be defined as follows:

Definition 2. *The* reputation *of a user is the expectation of its behavior based on other nodes' observations or on the collected information about the user's past behavior within a specific context at a given time.*

4.2 System Node Trust Model

The *trust level* of node p_j can provide security services, such as authentication (a), integrity (g), and confidentiality (d) for users, which is defined as follows:

$$\begin{cases} TL^k_{j,id} = \varepsilon_j \times H(j,id,k,t) + \beta_j \times G(id,k,t), \\ \qquad\qquad k \in (a,g,d) \\ \varepsilon_j + \beta_j = 1, \quad \varepsilon_j \geq 0, \quad \beta_j \geq 0. \end{cases} \tag{1}$$

The kth trustworthiness at a given time t between the node and the user, for example, node p_j for user id, denoted as $TL^k_{j,id}$, is computed based on the direct relationship at time t between node p_j and user id, denoted as $H(j,id,k,t)$; as well as the reputation of user id at time t is denoted as $G(id,k,t)$. Let the weights given to direct trust and reputation relationships be ε and β, respectively. The trust relationship is a function between the direct trust and the reputation. If the trustworthiness is based more on the direct trust relationship with p_j for id than the reputation of id, ε will be larger than β. Symbol k denotes the type of security provided by node p_j, such as authentication, integrity, and confidentiality.

The direct trust relationship is varied by service provider and user's direct interaction, environment changing, and the decay as the time going on.

- The direct trust of node p_j for user id will increase or decrease when p_j directly interacts with id. For example, if user id executes task on node p_j and p_j can carry out this task without any security threats, the trust of p_j for id will increase. Otherwise if p_j is attacked by user id, the direct trust of p_j for id will decrease.
- A dynamic change in available resource can affect the direct trust of p_j for id. For an instance, if node p_j has more idle storage resources, user id can use more storage resources, which means the task of user id can be completed successfully with higher probability. Thus, the trust of node p_j for id will increase. Otherwise, it decreases as node p_j has less available storage resources.
- As time going on, the direct trust itself decays.

Now, we build a direct trust differential equation that is derived from the brand image [39], [45] in economics. The brand image has been defined as the consumer's mental picture of the offering, and it includes symbolic meanings that consumers associate with the specific attributes of the product or service, and can be modeled as a differential game with an infinite time horizon [39], [45]. The direct trust can be computed as:

$$\begin{cases} H'(j,id,k,t) = \dfrac{dH}{dt} = \xi D(j,id,k,t) \\ \qquad\qquad + \varphi E(j,id,k,t) - \rho H(j,id,k,t), \\ H(j,id,k,0) = H_0(j,id,k). \end{cases} \tag{2}$$

The kth trust value for the direct interaction of node p_j with user id at a given time t is $D(j,id,k,t)$, and $E(j,id,k,t)$ represents the trust function between p_j and user id considering the environment changes. The parameters ξ, φ, and ρ are positive, although we shall consider $\xi = 0$, $\varphi = 0$ as a limited case and ρ presents the decay rate. The solution of (2) is

$$H(j,id,k,t) = H_0(j,id,k)e^{-\rho t} + \int_0^t [\xi D(j,id,k,s) + \varphi E(j,id,k,s)]e^{-\rho(t-s)}ds.$$

In economics, the user always tries his best to obtain a better brand image [39], [45]. In other words, every user ties to increase his trust level, which is the main contribution to his reputation. The other contribution of his reputation is the trust recommendations by others and itself decay rate. Hence, user id's kth reputation can be expressed as a following differential equation:

$$\begin{cases} G'(id,k,t) = \dfrac{dG}{dt} = \mu A(id,k,t) \\ \qquad\qquad + \nu \displaystyle\sum_{j \in (all\ nodes)} P(j,id,t) - \delta G(id,k,t), \\ G(id,k,0) = G_0(id,k). \end{cases} \tag{3}$$

The kth brand image trust function of user id is expressed as $A(id,k,t)$, and $P(j,id,t)$ is the trust recommendations by other system service provider nodes. As the recommendations trust is primarily based on what other nodes say about a node, we introduce the recommendation trust factor ν to prevent any cheating via collusion among a group of nodes. Hence, ν is a value between 0 and 1, and will have a higher value if the recommender does not have an alliance with the targeted user. In (3), μ is positive and δ is decay rate. The solution of (3) is

$$G(id,k,t) = G_0(id,k)e^{-\delta t} + \int_0^t \left[\mu A(id,k,s) + \nu \sum_{j \in (all\ nodes)} P(j,id,s)\right] e^{-(t-s)}ds.$$

Consider a system with three service provider nodes and two users in a HDS, the service provider parameters are listed in Table 3, and the parameters in our trust model are listed in Tables 4 and 5. Table 6 shows the results for this example normalized in range between 0 and 1. From the results, we observe that the two and three service provider nodes accept user 1 as a very trustworthy user with security service integrity, for the trust level is 0.9. We also conclude that the node 2 accepts user 2 as a very untrustworthy user with security service authentication, for the trust level is 0.1. These results will be used to compute the security overheads and risk rate in next section.

4.3 Security Overhead Model

We consider a class of HDSs in which an application is comprised of a collection of tasks to accomplish an overall mission. These tasks are dependent with each other and the application is represented by a DAG. Each task requires three security services (authentication, integrity, and confidentiality) with various security levels specified by the user. The values of these security levels are normalized in a

TABLE 4
The Function of Our Trust Model

notation	value	notation	value	notation	value	notation	value
$D(1,1,a,t)$	$t+0.5$	$E(1,1,a,t)$	$t^{1.5}+1$	$H_0(1,1,a)$	0.3	$A(1,a,t)$	t
$D(1,1,g,t)$	t^2+1	$E(1,1,g,t)$	$t+1$	$H_0(1,1,g)$	0.32	$A(1,g,t)$	$0.5t$
$D(1,1,d,t)$	$t+1$	$E(1,1,d,t)$	$0.5t$	$H_0(1,1,d)$	0.4	$A(1,d,t)$	$0.2t+1$
$D(2,1,a,t)$	$t+0.1$	$E(2,1,a,t)$	$1.5t+1$	$H_0(2,1,a)$	0.2	$A(2,a,t)$	$0.3t+1$
$D(2,1,g,t)$	$3t+0.2$	$E(2,1,g,t)$	$1.2t+1$	$H_0(2,1,g)$	0.23	$A(2,g,t)$	$t+1$
$D(2,1,d,t)$	$0.9t+1$	$E(2,1,d,t)$	t	$H_0(2,1,d)$	0.5	$A(2,d,t)$	$5t+1$
$D(3,1,a,t)$	$0.5t+1$	$E(3,1,a,t)$	t	$H_0(3,1,a)$	0.6	$P(1,1,t)$	$5t$
$D(3,1,g,t)$	$t+1.6$	$E(3,1,g,t)$	$0.7t$	$H_0(3,1,g)$	0.3	$P(3,1,t)$	$\cos(0.5t)$
$D(3,1,d,t)$	$3t+1$	$E(3,1,d,t)$	$0.8t+1$	$H_0(3,1,d)$	0.2	$P(1,2,t)$	$0.5t+1$
$D(1,2,a,t)$	$2t+1$	$E(1,2,a,t)$	$0.6t+1$	$H_0(1,2,a)$	0.3	$P(2,2,t)$	$sin(t+1)$
$D(1,2,g,t)$	$t+1$	$E(1,2,g,t)$	$2t+1$	$H_0(1,2,g)$	0.2	$G_0(1,a)$	0.7
$D(1,2,d,t)$	$t+1$	$E(1,2,d,t)$	$5t+1$	$H_0(1,2,d)$	0.22	$G_0(1,e)$	0.5
$D(2,2,a,t)$	$0.2t$	$E(2,2,a,t)$	t^3+1	$H_0(2,2,a)$	0.32	$G_0(1,g)$	0.4
$D(2,2,g,t)$	$5t+1$	$E(2,2,g,t)$	$2.5t$	$H_0(2,2,g)$	0.45	$G_0(2,a)$	0.4
$D(2,2,d,t)$	$10t+1$	$E(2,2,d,t)$	$t+0.1$	$H_0(2,2,d)$	0.44	$G_0(2,e)$	0.6
$D(3,2,a,t)$	$1.5t+1$	$E(3,2,a,t)$	$t+0.3$	$H_0(3,2,a)$	0.36	$G_0(2,g)$	0.8
$D(3,2,g,t)$	$1.5t+1$	$E(3,2,g,t)$	$t+1$	$H_0(3,2,g)$	0.6		
$D(3,2,d,t)$	$2.5t$	$E(3,2,d,t)$	t	$H_0(3,2,d)$	0.9		

range from 0 to 1 as well. For example, we can have a task specified with 0.7 for the authentication service, 0.8 for the integrity service, and 0.3 for the confidentiality service. Note that the same security level value in different security services may have various meanings. As $SD^k_{i,j}$ is the security level the task t_i requires including authentication, integrity, and confidentiality services provided by node p_j and is specified by the user. Let $C^k_{i,j}$ be the security overhead of the kth security service. Then, the security overhead $C^k_{i,j}$ experienced by t_i on node p_j can be computed by using (4). The overall security overhead of t_i on node p_j with security requirements for the three services above is modeled by (5).

TABLE 5
The Parameters of Our Trust Model

parameter	value	parameter	value	parameter	value
ε_1	0.3	β_1	0.7	ε_2	0.4
β_2	0.6	ε_3	0.6	β_3	0.4
ξ	0.1	φ	0.5	ρ	0.55
μ	0.6	ν	0.7	δ	0.12

TABLE 6
The Trust Level Results

trust type(k)	$TL_{1,1}$	$TL_{2,1}$	$TL_{3,1}$	$TL_{1,2}$	$TL_{2,2}$	$TL_{3,2}$
authentication(a)	0.8	0.8	0.7	0.6	0.1	0.7
integrity(g)	0.8	0.9	0.9	0.6	0.7	0.2
confidentiality(d)	0.3	0.5	0.9	0.8	0.5	0.9

$$C^k_{i,j}=\begin{cases}0, & if\ SD^k_{i,j}\le TL^k_{j,id},\\ S_k(SD^k_{i,j}-TL^k_{j,id})/u_j, & k\in(a,g,d)\\ & if\ SD^k_{i,j}>TL^k_{j,id},\end{cases}\tag{4}$$

$$\begin{cases}C_{i,j}=\sum\limits_{k\in(a,g,d)} w_k C^k_{i,j},\\ \sum\limits_{k\in(a,g,d)} w_k=1,\end{cases}\tag{5}$$

where trust level $TL^k_{j,id}$ of node p_j for user id can be calculated by (1) and normalized in the range [0, 1]. In our security overhead model, some techniques are used to ensure security service and lead to security overhead [26]. Such as DES cryptographic algorithm for confidentiality, HMAC-MD5 technique for authentication, RIPEMD-128 hash function for integrity, and so on. An example of such security overhead derived from literature [26] is shown in Table 7 and achieved by $S_k(x)$ function. In order to reflect the importance of three security services in overall security overhead $C_{i,j}$, we assign the weight w_k to three security services. For example, $w_a=0.6$, $w_g=0.3$, $w_d=0.1$ indicates that authentication is more important than integrity and confidentiality. The implications of the above relationships

TABLE 7
Authentication Methods

Authentication Methods	$S_a(x)$:Security Level	Computation Time(ms)
HMAC-MD5	0.55	90
HMAC-SHA-1	0.91	148
CBC-MAC-AES	1	163

are reasonable, since a security mechanism providing higher security service imposes higher overheads than the mechanisms offering lower security service.

4.4 Task Security Analysis

Since many parallel applications operate in HDSs that are not risk-free during the course of the execution, it is necessary and important for these systems to have security awareness. In other words, the system can quantitatively evaluate the level of security. Therefore, we derive the risk probability to quantitatively analyze the risk rate for a task scheduled on node p_j.

In this security analysis model, we assume that the risk rate is a function of security levels and the distribution of the risk for any fixed time interval follows a $Poisson$ probability distribution. The risk rate model is used for illustration purpose only. Thus, the task's risk rate of the kth security service on a specific node can be replaced by an exponential distribution as follows:

$$Pr(t_i^k, p_j) = \begin{cases} 0, \ if \ SD_{i,j}^k \le TL_{j,id}^k, \\ 1 - e^{-\lambda_j^k (SD_{i,j}^k - TL_{j,id}^k)}, \\ \quad if \ SD_{i,j}^k > TL_{j,id}^k. \end{cases} \quad k \in (a,g,d) \tag{6}$$

In heterogeneous systems, the risk coefficient λ_j^k is different from one to another. For example, node p_j can provide authentication with coefficient equals 1.3, integrity coefficient equals 3.2, and confidentiality coefficient equals 0.3. While the other node p_x may offer values 2.3, 0.1, 0.9, and 3.3, respectively. The negative exponent indicates that failure probability grows with the difference $SD_{i,j}^k - TL_{j,id}^k$. The task abortion at a node could result from severe network attack or inaccessibility from a security imposed barricade. The rate of task t_i on node p_j can be obtained below by considering all the security services provided to this task. Consequently, we have

$$Pr(t_i, p_j) = 1 - \prod_{k \in (a,g,d)} (1 - Pr(t_i^k, p_j)). \tag{7}$$

The task risk probability $Pr(t_i)$, when task t_i is assigned to node p_j, is $Pr(t_i, p_j)$. Given a task set T, the system risk probability $Pr(T)$ that all tasks are free from being attacked during their executions is computed based on (8). Thus, we have

$$Pr(t_i) = \sum_{j=1}^{m} q_{i,j} Pr(t_i, p_j)$$

$$q_{i,j} = \begin{cases} 1, & if \ t_i \ assign \ to \ p_j, \\ 0, & other, \end{cases} \tag{8}$$

$$Pr(T) = 1 - \prod_{t_i \in T} (1 - Pr(t_i)), \tag{9}$$

where $q_{i,j}$ denote task t_i is assigned to service provider node p_j. The task risk probability $Pr(t_i, p_j)$ will be used to search the processor set P' with higher security, and system risk probability $Pr(T)$ will also be used as a metric to evaluate our security-driven scheduling algorithm in next section.

TABLE 8
The $SRank$ Value of Each Task

Task	t_1	t_2	t_3	t_4	t_5	t_6	t_7
$SRank$	52.608	37.714	39.797	32.664	14.266	17.662	6.321

5 Security-Driven List Scheduling Algorithm

List scheduling basically consists of two phases: a task prioritization phase, wherein a certain priority is computed and assigned to each task in DAG; The second phase is the machine assignment, wherein each task (in order of its priority) is assigned to a machine that minimizes the cost function. In this section, first we outline the concept of security-aware task priority. Then, we propose and analyze the security-driven list scheduling algorithm (**SDS**).

5.1 Task Priorities

Our security-driven scheduling algorithm will use security upward rank ($SRank$) attribution to compute tasks priorities. The $SRank$ is explained in Definition 3.

Definition 3. *Given a DAG with v tasks and e edges and a system with m heterogeneous processors, the $SRank$ during a particular scheduling step is a rank of task, from an exit task to itself, which has the sum of communication costs of edges, computational costs, and security overheads of tasks over all processors. Communication costs between tasks scheduled on the same processor are assumed to be zero, however, the execution constraints are preserved.*

The $SRank$ is recursively defined as follows:

$$\begin{cases} SRank(t_i) = \left(\sum_{j=1}^{m} SRank(t_i, p_j) \right) \Big/ m \\ \qquad + max_{t_x \in succ(t_i)} (w(e_{i,x}) + SRank(t_x)), \\ SRank(t_i, p_j) = w(t_i)/u_j + C_{i,j}, \end{cases} \tag{10}$$

where $SRank(t_i, p_j)$ is the security upward rank of task t_i on node p_j, $C_{i,j}$ is the security overhead computed by (5) and $succ(t_i)$ is the set of immediate successors of task t_i. As the communication speed is set to 1, the communication cost of edge $e_{i,x}$ equals to its weight. The rank is computed recursively by traversing the task graph upward, starting from the exit task to entry task. Since we recursively compute the $SRank$ values of the tasks in a given DAG upward starting for the exit task (hence, the name "upward rank"), the task with the highest $SRank$ value will always be entry task t_{entry}. For the exit task t_{exit}, the rank value is equal to

$$SRank(t_{exit}) = \left(\sum_{j=1}^{m} (w(t_{exit})/u_j + C_{exit,j}) \right) \Big/ m. \tag{11}$$

For example, consider the application DAG in Fig. 2a, which is submitted by user $id = 1$, and their security demand are listed in Table 2. The target HDS architecture and computational parameters are shown in Fig. 2b and Table 3. The task $SRank$ value, which is recursively computed by (10), (11), is shown in Table 8.

5.2 Design of Security-Driven List Scheduling Algorithm

Now we shall present our SDS algorithm, which incorporates the security requirements into scheduling for HDSs. Before presenting the objective function, it is necessary to define the Earliest Start Time (EST) and Earliest Finish Time (EFT) attributes, which are derived from a given partial schedule. $EST(t_i, p_j)$ and $EFT(t_i, p_j)$ are the earliest start execution time and the earliest execution finish time of task t_i on processor node p_j, respectively. For the entry task t_{entry}, the EST is defined as

$$EST(t_{entry}, p_j) = 0. \tag{12}$$

For the other tasks in DAG graph, EST and EFT values are computed recursively, starting from the $entry$ task, as shown in (13) and (14), respectively. In order to compute EST of a task t_i, all immediate predecessor tasks t_x of t_i must be scheduled.

$$EST(t_i, p_j) = max\{Available(t_i, p_j), \\ max_{t_x \in pred(t_i)}(EFT(t_x, p_n) + g_{x,i})\}, \tag{13}$$

$$EFT(t_i, p_j) = EST(t_i, p_j) + w(t_i)/u_j + C_{i,j}, \tag{14}$$

where $g_{x,i}$ is the communication cost of edge $e_{x,i}$ transferring data from task t_x (scheduled on p_n) to task t_i (scheduled on p_j), these data are equal to this edge weight $w(e_{x,i})$. When both t_x and t_i are scheduled on the same processor, $g_{x,i}$ becomes zero, since we assumed that the intraprocessor communication cost is negligible when it is compared with the interprocessor communication cost. The $pred(t_i)$ is the set of immediate predecessor tasks to task t_i, and $Available(t_i, p_j)$ is the earliest time at which processor p_j is ready for task execution. Using insertion-based policy [18], the time between $EST(t_i, p_j)$ and $EFT(t_i, p_j)$ is also available. The earliest inner block in the EST equation returns the ready time, i.e., the time instance at which all the data needed by t_i has arrived at processor p_j. After all the tasks in the graph are scheduled, the schedule length (i.e, overall completion time) will be the actual finish time of $exit$ task t_{exit}, thus the schedule length (which is also called makespan) is defined as follows:

$$makespan = EFT(t_{exit}, p_j). \tag{15}$$

The proposed SDS algorithm is outlined in Algorithm 1. The goal of this algorithm is to deliver the task with the minimum risk-free probability and maintains high performance for tasks running on HDSs. To achieve this goal, SDS arranges task scheduling sequence by $SRank$ that takes security overheads into account. For each task, SDS computes $EFT(t_i, p_j)$ by using (13), (14), and computes risk probability $Pr(t_i, p_j)$ by using (7). These ideas are implemented from steps 5 to 8. In step 9, we try to find a set P' that their risk probability $Pr(t_i, p_j)$ is less than a constant θ. At last, SDS finds a node with the earliest execution finish time at step 10 in Algorithm 1 and assigns task t_i to this node.

5.3 Algorithm Complexity Analysis

The time complexity of scheduling algorithms for parallel application DAG is usually defined in terms of the number of tasks v, the number of edges e, and the number of nodes m. The time-complexity analysis of our algorithm is in the following:

- There are v tasks in a parallel application DAG. For each task t_i, the computation of $SRank$ use (10) can be done in time $O(m + d_{max})$, where d_{max} is the maximum in/out degree of a task in the DAG. Hence, computing $SRank$ of the tasks can be done in time $O(v(m + d_{max}))$.
- Sorting the tasks by $SRank$ can be done in time $O(vlogv)$.
- For steps 5-8, computing the earliest finish time and the task risk probability to service provider node p_j use (13), (14), and (7) have complexity of $O(d_{max})$, and $O(4)$, respectively. In general case, $d_{max} \geq 4$. Hence, these steps can be done in time $O(md_{max})$.
- In the worst case, finding the set P' and the optimal node p_j in P' are all done in time $O(m)$. As steps 3-11 schedule all tasks in parallel application DAG, the complexity of that is $O(v(md_{max} + m + m)) = O(vm(d_{max} + 2))$.

Hence, the time complexity for SDS algorithm is

$$O(v(m + d_{max}) + vlogv + vm(d_{max} + 2)) \\ = O(max\{vlogv, vm(d_{max} + 2)\}).$$

```
1  Compute the SRank for all tasks by traversing
   graph from the exit task
2  Sort the tasks into a scheduling list by
   non-increasing order of SRank
3  while the scheduling list is not empty do
4      Remove the first task t_i from the scheduling list
5      for each node p_j ∈ P do
6          Compute EFT(t_i, p_j) using Eq.(13), Eq.(14)
7          Compute Pr(t_i, p_j) using Eq.(7)
8      end
9      Search the node set P' with Pr(t_i, p_j) < θ
10     Assign task t_i to the node p_j ∈ P' that minimize
       EFT of t_i
11 end
```

Algorithm 1. The proposed SDS algorithm.

6 Performance Evaluation

In this section, we compare the performance of SDS algorithm with two well-known scheduling algorithms in HDSs: HEFT [13] and DLS [10] algorithms. In the first experiment, we compare with original HEFT and DLS. Then, to make the comparisons fair, we attempt to slightly modify the two algorithms in such a way that they can arbitrarily pick a security level from the security-level range for each service required by a task. Although these algorithms are intended to schedule tasks with security requirements, they make no effort to optimize the quality of security.

To test the performance of these algorithms, we have built an extensive simulation environment of HDSs with 64 processors that computation capacities varies from Pentium II to Pentium IV. The heterogeneous processors are connected via a fully connected network, which the

transmission rates of links are assumed to be 100 Mbits/second. Parallel application graphs are based on random generation by varying parameters such as CCR, *Parallelism factor* α (see Section 6.1) and some of the numerical real-life problems. The following experiments are executed based on the security-driven scheduling architecture, which is designed in Section 3.1.

The performance metrics chosen for the comparison are the makespan (see (15)), task risk probability (see (8) and (9)) and the speedup that is computed by dividing the sequential execution time (i.e., cumulative computation costs and security overheads of the tasks in the graph) by the parallel execution time (i.e., the makespan of the output schedule) as shown in (16). The sequential execution time is computed by assigning all tasks to a single processor that minimizes the cumulative of the computation costs and security overheads. If the sum of the computational costs and security overheads is maximized, it results a higher speedup, but ends up with the same rank of the scheduling algorithms. The comparison is intended not only to present quantitative results, but also to qualitatively analyze the results and to suggest the reasons, for a better understanding of the overall scheduling problem.

$$Speedup = \frac{min_{u_j \in U}\{\sum_{t_i \in T}(w(t_i)/u_j + C_{i,j})\}}{makespan}. \quad (16)$$

6.1 Randomly Generated Application Graphs

In our study, we first considered the randomly generated application graphs. Such that a random graph generator is implemented to generate weighted application DAGs with various characteristics that depend on several input parameters given below. Our simulation-based framework allows assigning sets of values to the parameters used by the random graph generator. For the generation of random graphs, which are commonly used to compare scheduling algorithms [13], [14], [40], [41] five fundamental characteristics of DAG are considered:

- *DAG size*, v: The number of tasks in the application DAG.
- *Communication to computational ratio, CCR*: The average communication cost divided by the average computation cost of the application DAG.
- *Parallelism factor*, α [13]: The number of levels of the application DAG is calculated by randomly generating a number, using a uniform distribution with a mean value of $\frac{\sqrt{v}}{\alpha}$, and then rounding it up to the nearest integer. The width of each level is calculated by randomly generating a number using a uniform distribution with a mean value of $\alpha \times \sqrt{v}$ and then rounding it up to the nearest integer [13]. A dense graph (a shorter graph with high parallelism) can be generated by selecting $\alpha > 1.0$ and a low α value leads to a DAG with a low parallelism degree [42].
- *out_degree*: Out degree of a node.
- *Computation cost heterogeneity factor, h* [13]: Higher h value indicates higher variance for the computation costs of a task, with respect to the service provider nodes in the system, and vice versa. If the heterogeneity factor is set to 0, the computation cost of a task is the same for all nodes. The average computation cost of a task $\overline{w(t_i)}$ is randomly generated using a uniform distribution generator with a mean value of W. The value of W does not affect the performance results of the scheduling algorithms. If there are m nodes in a HDSs, the computation cost of a task t_i for each node is set by randomly selecting m computation cost values of t_i from the range $[\overline{w(t_i)} \times (1 - \frac{h}{2}), \overline{w(t_i)} \times (1 + \frac{h}{2})]$. The m selected computation cost values of t_i are sorted in an increasing order. The computation cost value of t_i on node p_1 is set to the first (i.e., lowest) computation cost. The computation cost of t_i on node p_2 is set to the second value. This allocation continues until all nodes are processed.

In our simulation experiments, graphs are generated for all combinations of the above parameters with number of tasks ranged between 400 and 2,000 with 400 steps. Every possible edge (DAGs are acyclic) is created with the same probability, which is calculated based on the average number of edges per task node. To obtain the desired CCR for a graph, task weights are generated randomly from a uniform distribution within the range of $[0.1, 1.9]$, by varying computation cost heterogeneity factor h, and thus the average task weight is 1. Edge weights are also taken from a uniform distribution with a mean depends on CCR and Parallelism factor α. The relative deviation of the edge weights is identical to that of the node weights. The size of the security-required data generated by a task is arbitrarily chosen from a triangular distribution. Every set of the above parameters are used to generate several random graphs in order to avoid scattering effects. The results presented below are the average of the results obtained for these graphs.

6.2 Experimental Results

The goal of the experiments is to compare the proposed SDS algorithm with the other two algorithms, HEFT and DLS. To stress the evaluation, we assume that each task arriving at system requires all of the three security services. In the first set of experiments, we assume the risk probability constant $\theta = 0.3$ in SDS algorithm. Next, we will examine the performance by various risk probability constant θ.

Fig. 3 shows the simulation results for the three algorithms (SDS, original HEFT, and original DLS) on the HDS that has been proposed above. We observe from Fig. 3a that SDS significantly outperforms the other two algorithms in terms of the makespan. We attribute the performance improvement of SDS over original HEFT and DLS to the fact that SDS is a security-adaptive scheduler and assigns a task to a service provider node not only considering its computational time but also its security demands. However, original HEFT and DLS schedule tasks only considering computational time and the success of task execution is inevitable need the security overhead on HDSs. Thus, the makespan of original HEFT and DLS is longer than SDS.

Fig. 3b plots the risk probability of the three algorithms when the tasks of DAG are increased from 400 to 2,000. Fig. 3b reveals that SDS consistently performs better, with respect to the quality of risk probability, than all the rest approaches, whereas original HEFT and DLS algorithms exhibit similar

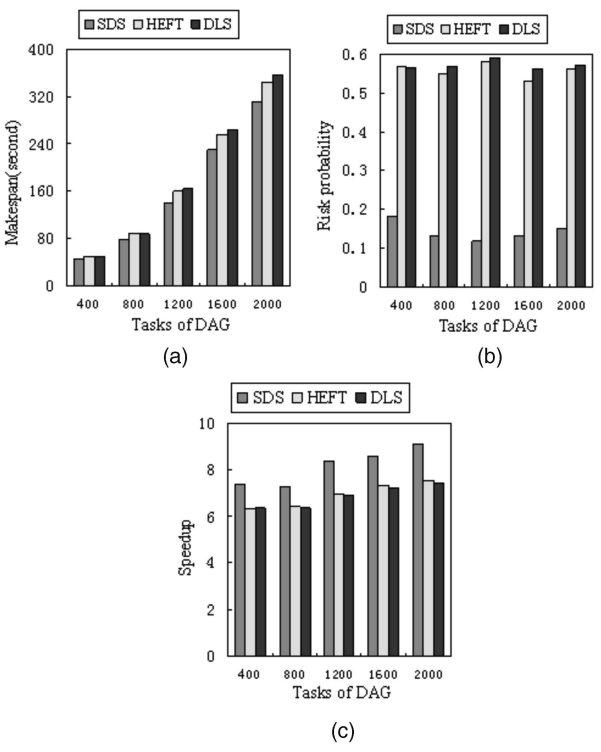


Fig. 3. Performance impact of tasks of DAG with original HEFT and DLS. (a) makespan in seconds; (b) risk probability; (c) speedup.

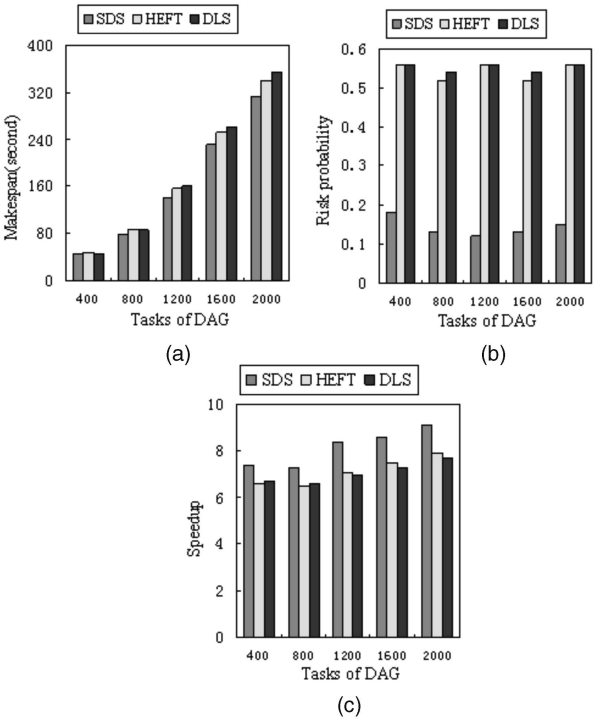


Fig. 4. Performance impact of tasks of DAG with modified HEFT and DLS. (a) makespan in seconds; (b) risk probability; (c) speedup.

performance. Specifically, SDS outperforms original HEFT, original DLS for risk probability by averages of 251.2 and 254.3 percent, respectively. This improvement is due to the fact that SDS approach is capable of employing the risk probability attribute to improve the quality of scheduling. Whereas, original HEFT and DLS natures are nonsecurity awareness, which merely select a service provider node for a task without considering the task's security demands. Another improvement with the speedup could be concluded from Fig. 3c.

The simulation experimental results of SDS with modified HEFT and DLS are shown in Fig. 4. From Fig. 4, we can get the similar conclusion of the above experiments. Thus, in the following experiments, we only consider the SDS, modified HEFT, and modified DLS. The makespan produced by SDS, HEFT, and DLS algorithms for the various CCR values from 0.1 to 2 are shown in Fig. 5a. The average makespan value of SDS algorithm is shorter than HEFT and DLS algorithms by: (11.5 percent, 11.3 percent), (10.3 percent, 10.5 percent), (8.4 percent, 8.3 percent), (5.7 percent, 4.7 percent) and (1.8 percent, 0.9 percent), for CCR is 0.1, 0.4, 0.8, 1.0, and 2.0, respectively. The first value of each parenthesized pair is the improvement achieved by SDS algorithm over HEFT algorithm, while the second value is the improvement of SDS over HEFT algorithm. The speedup values achieved by the three algorithms with respect to certain CCR values are shown in Fig. 5c. The average speedup value of SDS algorithm is higher than those returned by HEFT and DLS algorithms by: (21.9 percent, 21.4 percent), (20.7 percent, 20.6 percent), (17.0 percent, 16.9 percent), (8.2 percent, 8.1 percent) and (2.3 percent, 2.4 percent), when the CCR is equal to: 0.1, 0.4, 0.8, 1.0, and 2.0, respectively. As the value of CCR increases, the interprocessor communication overheads dominate the computation and hence, the performance of SDS improved compared with HEFT, DLS which tends to degrade. Thus, SDS algorithm is suitable for security-sensitive application with high computation demands. Whereas, the risk probability gained by varying CCR is similar to varying tasks of DAG, and also could be concluded from Fig. 5b.

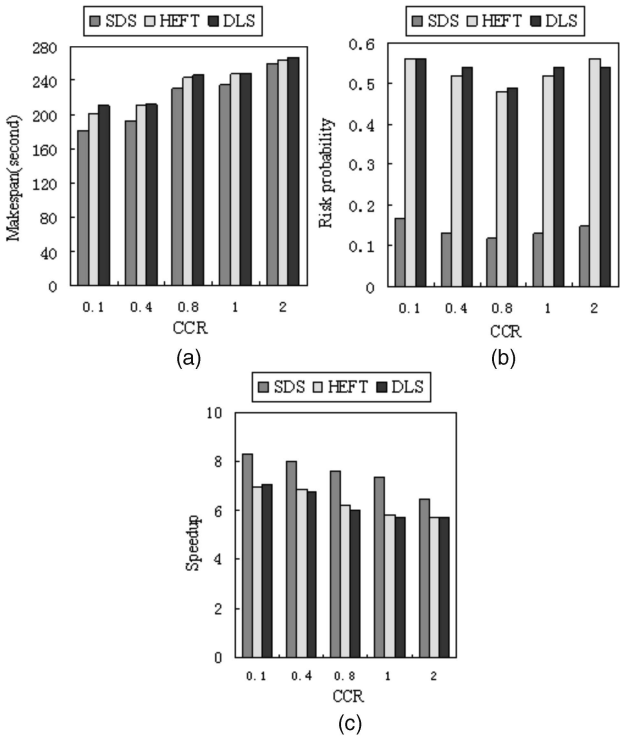


Fig. 5. Performance impact of CCR. (a) makespan in seconds; (b) risk probability; (c) speedup.

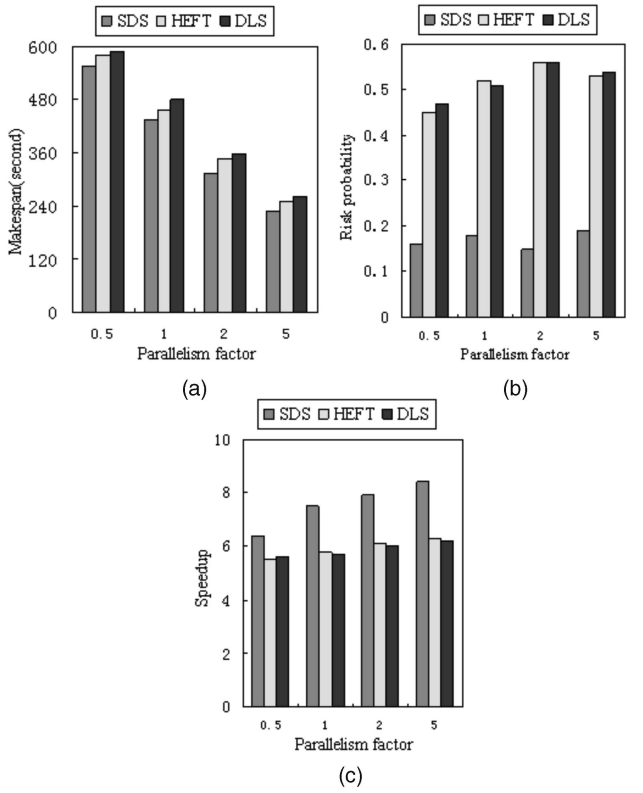


Fig. 6. Performance impact of parallelism factor α. (a) makespan in seconds; (b) risk probability; (c) speedup.

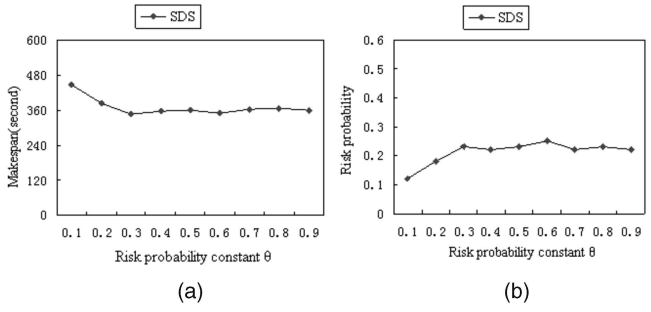


Fig. 7. Performance impact of risk probability constant θ. (a) makespan in seconds; (b) risk probability.

To examine the performance sensitivity for the three algorithms to the parallelism factor α, in this set of experiments, we vary α from 0.5 to 5. The results reported in Fig. 6 reveal that SDS outperforms the other two in terms of makespan, risk probability, and speedup. As the parallelism factor α increases, the improvement becomes more significant. This is mainly because high parallel application can be exploited by scheduling algorithm.

The selection of suitable risk probability constant θ is crucial to an ultimate performance for the security-driven scheduling algorithm. To illustrate the effects of using different risk probability constant, we carry out additional experiments by varying the risk probability constant θ. Fig. 7 plots the makespan and risk probability results from using SDS algorithms to schedule 2,000 tasks. The results indicate that a small risk probability constant θ does not lead to a better performance. In other words, the risk probability constant θ cannot be made too small. Indeed, if we increase θ after $\theta = 3$, the performance improvement will not be significant. In practice, we find the near-optimal choice when $\theta = 0.3$ at the inflection point of the plotted curves. It is interesting to see $\theta = 0.3$ is optimal for both makespan and risk probability in our approach.

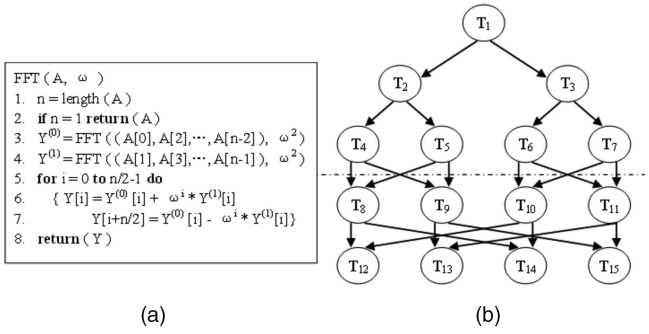


Fig. 8. (a) FFT algorithm; (b) the DAG generated by FFT with four points.

6.3 Performance Analysis on Application Graphs of Real-Life Problems

Using real applications to test the performance of algorithms is very common [13], [30], [44]. In addition to randomly generated DAGs, we also simulate a real-life problems: Fast Fourier Transformation [13], [44].

The recursive, one-dimensional FFT algorithm [13], [44] and its task graph (when there are four data points) is given in Fig. 8. In this figure, A and Y are arrays, $A[0 : n-2 : 2] = \{A[0], A[2], \ldots, A[n-2]\}$, and $A[1 : n-1 : 2] = \{A[1], A[3], \ldots, A[n-1]\}$. The behavior of FFT with input vector size $= 4$ is shown in Fig. 8b. The computation of FFT consists of two operations, the input vector operation (IVO) (lines 3 and 4 in FFT algorithm) and the butterfly operation (BO) (lines 5-8 in FFT algorithm). The task graph in Fig. 8b can also be divided into two parts, the tasks above the dashed line are the recursive call task (IVO) and the others below the line are the butterfly operation tasks. Since the grain size of a DAG determines the value of CCR, to produce the desire value of CCR we want, array A can be split into appropriate number of subarrays.

DAG of FFT used for the FFT application here has $2^l - 1$ IVO-task, 2^l FFT-task, and $(l+1) \times 2^l$ BO-task, where $l > 0$. For each IVO-task with input vector size $= k$, it is required to send a vector with size $= k/2$ to its immediate successors. For each FFT-task or BO-task with input vector size $= k$, it is required to send a vector with size $= k$ to its immediate successors. We found out if the number of FFT-task is less than or equal to 16 and array A has 1,024 elements, we have $CCR < 0.5$. If the number of FFT-task is equal to 32 and array A has 1,024 elements, we have $0.5 \leq CCR \leq 1$. If the number of FFT-task is greater than or equal to 64 and array A has 1,024 elements, we have $CCR > 1$. According to the values of CCR, we generate DAGs with different number of tasks. In this experiment, the input vector size of FFT is 1,024, and the size of security-required data for each FFT task is arbitrarily chosen from a triangular distribution. Fig. 9 demonstrates that SDS can achieve noticeable performance improvements for the real-life applications.

7 Conclusions and Future Work

In this paper, we attempt to incorporate the security awareness into task scheduling in HDSs. We believe that it is mandatory to design and implement security-driven

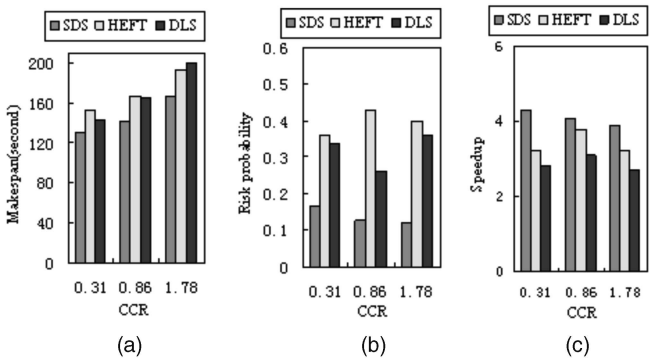


Fig. 9. Performance impact of FFT. (a) makespan in seconds; (b) risk probability; (c) speedup.

scheduling to meet the security requirements of task dependency applications while achieving good performances. Without security-driven scheduler, the following two situations may occur. First, security-sensitive applications will run at a lower security levels, thereby leading to low quality of security. Second, security-sensitive applications will be at a higher security levels with higher security overheads, which can result in poor performance. These two scenarios could happen because nonsecurity-driven schedulers do not take security overheads into account while making scheduling decisions. To solve this problem, we had built a security-driven scheduling architecture that can dynamically measure the *trust level* of service provider node by differential equations and integrated the security into scheduling. Then, we evaluated the security overheads of tasks and analyzed the risk probability of the system. At last, we had designed a security-driven scheduling algorithm, SDS, including the computation of the scheduling priorities by *SRank* that take the security overheads into account. The main idea of our proposed algorithm is to incorporate the security overheads and the risk probability into scheduling.

The performance of Security-Driven Scheduling algorithm was compared with two of the best existing scheduling algorithms for HDSs: HEFT and DLS algorithms. The comparisons were based on both randomly generated application DAGs and a real-life problem fast fourier transformation (FFT). The experimental results demonstrate that SDS algorithm significantly outperforms both HEFT and DLS algorithms in terms of scheduling length (makespan), risk probability, and speedup.

This work is one of the first attempts to investigate the HDS with security implications and carefully design a scheduler that absorbs security measures in executing a task. Future studies in this domain are twofold: first it will be interesting to extend our security overhead models to multidimensional computing resources, such as network bandwidth, memory, and storage; second, it will be interesting to study the problem context under an arbitrarily interconnected HDS with security requirements.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable comments and suggestions. This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 90715029 and 60873074), the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (Grant No. 708066), the Program for New Century Excellent Talents in University, NCET, and a Project supported by Scientific Research Fund of Hunan Provincial Education Department (08C435, 09A043).

REFERENCES

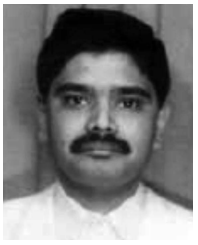
[1] A.A. Alhamdan, "Scheduling Methods for Efficient Utilization of Clusters Computing Environments," PhD thesis, Univ. of Connecticut, 2003.
[2] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, 1999.
[3] J. Sarangapani, *Wireless Ad Hoc and Sensor Networks: Protocols, Performance, and Control.* CRC Press, Apr. 2007.
[4] A. Dogan and F. Özgüner, "LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. Int'l Conf. Parallel Processing,* pp. 352-359, 2002.
[5] G. Donoho, "Building a Web Service to Provide Real-Time Stock Quotes," MCAD.Net, Feb. 2004.
[6] 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.
[7] W. Yurcik, X. Meng, G. Koenig, and J. Greenseid, "Cluster Security as a Unique Problem with Emergent Properties," *Proc. Fifth LCI Int'l Conf. Linux Clusters: The HPC Revolution,* May 2004.
[8] K. Connelly and A.A. Chien, "Breaking the Barriers: High Performance Security for High Performance Computing," *Proc. Workshop New Security Paradigms,* Sept. 2002.
[9] H. El-Rewini and T.G. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *J. Parallel and Distributed Computing,* vol. 9, no. 2, pp. 138-153, 1990.
[10] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Machine Architectures," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 2, pp. 175-187, Feb. 1993.
[11] M. Iverson, F. Ozuner, and G. Follen, "Parallelizing Existing Applications in a Distributed Heterogeneous Environment," *Proc. Heterogeneous Computing Workshop,* pp. 93-100, 1995.
[12] Y.-K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs onto Multiprocessors," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 5, pp. 506-521, May 1996.
[13] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 260-274, Mar. 2002.
[14] X. Tang, L. Kenli, and D. Padua, "Communication Contention in APN List Scheduling Algorithm," *Science in China Series F: Information Sciences,* vol. 52, no. 1, pp. 59-69, 2009.
[15] M.R. Gary and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.
[16] I. Ahmad and Y.-K. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 9, pp. 872-892, Sept. 1998.
[17] S. Darbha and D.P. Agrawal, "Optimal Scheduling Algorithm for Distributed-Memory Machines," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 1, pp. 87-95, Jan. 1998.
[18] C.H. Papadimitriou and M. Yannakakis, "Towards An Architecture-Independent Analysis of Parallel Algorithms," *SIAM J. Computing,* vol. 19, no. 2, pp. 322-328, 1990.
[19] M.K. Dhodhi, I. Ahmad, A. Yatama, and I. Ahmad, "An Integrated Technique for Task Matching and Scheduling onto Distributed Heterogeneous Computing System," *J. Parallel and Distributed Computing,* vol. 62, no. 9, pp. 1338-1361, 2002.
[20] H. Topcuoglu, S. Hariri, and M.Y. Wu, "Task Scheduling Algorithms for Heterogeneous Machines," *Proc. Heterogeneous Computing Workshop,* pp. 3-14, 1999.
[21] A. Radulescu and A.J.C. van Gemund, "Low-Cost Task Scheduling for Distributed-Memory Machines," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 6, pp. 648-658, June 2002.
[22] H.J. Park and B.K. Kim, "An Optimal Scheduling Algorithm for Minimizing the Computing Period of Cyclic Synchronous Tasks on Multiprocessors," *J. Systems and Software,* vol. 56, no. 3, pp. 213-229, 2001.

[23] D. Kim and B.G. Yi, "A Two-Pass Scheduling Algorithm for Parallel Programs," *Parallel Computing,* vol. 20, no. 6, pp. 869-885, 1994.
[24] A. Dogan and F. Özgüner, "On QoS-Based Scheduling of a Meta-Task with Multiple QoS Demands in Heterogeneous Computing," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 50-55, 2002.
[25] T. Xie, X. Qin, and A. Sung, "SAREC: A Security-Aware Scheduling Strategy for Real-Time Application on Clusters," *Proc. Int'l Conf. Parallel Processing (ICPP '05),* pp. 5-12, 2005.
[26] T. Xie and X. Qin, "Scheduling Security-Critical Real-Time Applications on Clusters," *IEEE Trans. Computers,* vol. 55, no. 7, pp. 864-879, July 2006.
[27] T. Xie, X. Qin, A. Sung, M. Lin, and L.T. Yang, "Real-Time Scheduling with Quality of Security Constraints," *Int'l J. High Performance Computing and Networking,* vol. 4, nos. 3/4, pp. 188-197, 2006.
[28] T. Xie and X. Qin, "Performance Evaluation of a New Scheduling Algorithm for Distributed Systems with Security Heterogeneity," *J. Parallel and Distributed Computing,* vol. 67, no. 6, pp. 1067-1081, 2007.
[29] T. Xie and X. Qin, "Security-Aware Resource Allocation for Real-Time Parallel Jobs on Homogeneous and Heterogeneous Clusters," *IEEE Trans. Parallel and Distributed Systems,* vol. 19, no. 5, pp. 682-697, May 2008.
[30] F. Azzedin and M. Maheswaran, "A Trust Brokering System and Its Application to Resource Management in Public-Resource Grids," *Proc. Parallel and Distributed Processing Symp.,* pp. 26-30, 2004.
[31] F. Azzedin and M. Maheswaran, "Integrating Trust into Grid Resource Management Systems," *Int'l Conf. Parallel Processing (ICPP '02),* pp. 47-54, 2002.
[32] R. He, J.W. Niu, and G.W. Zhang, "CBTM: A Trust Model with Uncertainty Quantification and Reasoning for Pervasive Computing," *Lecture Notes in Computer Science,* pp. 541-552, Springer-Verlag, 2005.
[33] M. Nielsen, K. Krukow, and V. Sassone, "A Bayesian Model for Event-Based Trust," *Electronic Notes in Theoretical Computer Science,* vol. 172, no. 1, pp. 499-521, 2007.
[34] S. Song, Y.-K. Kwok, and K. Hwang, "Risk-Resilient Heuristics and Genetic Algorithms for Security-Assured Grid Job Scheduling," *IEEE Trans. Computers,* vol. 55, no. 6, pp. 703-719, June 2006.
[35] F.G. Marmol and G.M. Peréz, "Security Threats Scenarios in Trust and Reputation Models for Distributed Systems," *Computers & Security,* vol. 28, no. 7, pp. 545-556, 2009.
[36] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks," *Proc. 12th Int'l Conf. World Wide Web,* 2003.
[37] S.A. Mokhov, "Towards Security Hardening of Scientific Demand-Driven and Pipelined Distributed Computing Systems," *Proc. Int'l Symp. Parallel and Distributed Computing (ISPDC '08),* pp. 375-382, 2008.
[38] M. Pourzandi, D. Gordon, W. Yurcik, and G.A. Koenig, "Clusters and Security: Distributed Security for Distributed Systems," *Proc. Fifth IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid '05),* pp. 96-104, 2005.
[39] S. Jorgensen, S. Taboubi, and G. Zaccour, "Retail Promotions with Negative Brand Image Effects: Is Cooperation Possible?" *European J. Operational Research,* vol. 150, no. 2, pp. 395-405, 2003.
[40] G.Q. Liu, K.L. Poh, and M. Xie, "Iterative List Scheduling for Heterogeneous Computing," *J. Parallel and Distributed Computing,* vol. 65, no. 5, pp. 654-665, 2005.
[41] M.I. Daoud and N. Kharma, "A High Performance Algorithm for Static Task Scheduling in Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing,* vol. 68, no. 4, pp. 399-409, 2008.
[42] S. Bansal, P. Kumar, and K. Singh, "An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 6, pp. 533-544, June 2003.
[43] C. Zhu, X. Tang, and K. Li, "Integrating Trust into Grid Economic Model Scheduling Algorithm," *Proc. Int'l Conf. Grid Computing, High-Performance and Distributed Applications (GADA '06),* pp. 1263-1272, 2006.
[44] Y. Chung and S. Ranka, "Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors," *Proc. Super computing,* pp. 512-521, 1992.
[45] A.E. Cretu and R.J. Brodie, "The Influence of Brand Image and Company Reputation Where Manufacturers Market to Small Firms: A Customer Value Perspective," *Industrial Marketing Management,* vol. 36, no. 2, pp. 230-240, 2007.
[46] F. Almenarez, A. Marin, C. Campo, and R.C. Garcia, "TrustAC: Trust-Based Access Control for Pervasive Devices," *Lecture Notes in Computer Science,* pp. 225-238, Springer-Verlag, 2005.
[47] H. Jameel, L.X. Hung, U. Kalim, A. Sajjad, S.Y. Lee, and Y.K. Lee, "A Trust Model for Ubiquitous Systems Based on Vectors of Trust Values," *Proc. Seventh IEEE Int'l Symp. Multimedia,* pp. 674-679, 2005.
[48] G. Theodorakopoulos and J.S. Baras, "On Trust Models and Trust Evaluation Metrics for Ad-Hoc Networks," *IEEE J. Selected Areas in Comm.,* vol. 24, no. 2, pp. 318-328, Feb. 2006.

Xiaoyong Tang received the master's degree from Hunan University, China, in 2007. He is currently working toward the PhD degree at Hunan University of China. His research interests include modeling and scheduling for distributed computing systems, distributed system reliability, distributed system security, and parallel algorithms.

Kenli Li received the BS degree in mathematics from Central South University, China, in 2000 and the PhD degree in computer science from Huazhong University of Science and Technology, China, in 2003. He has been a visiting scholar at the University of Illinois at Champaign and Urbana from 2004 to 2005. He is now a professor of computer science and technology at Hunan University. He is a senior member of the CCF. His major research contains parallel computing, grid and cloud computing, and DNA computer.

Zeng Zeng received the BS and MS degrees in automatic control from Huazhong University of Science and Technology, Wuhan, P.R. China, in 1997 and 2000, respectively, and the PhD degree in electrical and computer engineering from the National University of Singapore, in 2005. From 2005 to 2008, he worked as a research fellow at the National University of Singapore. His research interests include distributed/grid computing systems, multimedia storage systems, wireless sensor networks, and controller area networks. Currently, he works as an associate professor in Hunan University, P.R. China.



Bharadwaj Veeravalli received the BSc degree in physics from Madurai Kamaraj Uiversity, India, in 1987, the master's degree in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1991 and the PhD degree from the Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India, in 1994. He did his postdoctoral research in the Department of Computer Science, Concordia University, Montreal, Canada, in 1996. He is currently with the Department of Electrical and Computer Engineering at the National University of Singapore, as a tenured associate professor. His main stream research interests include multiprocessor systems, cluster/grid computing, scheduling in parallel and distributed systems, bioinformatics and computational biology, and multimedia computing. He is one of the earliest researchers in the field of divisible load theory (DLT). He is currently serving on the Editorial Board of the *IEEE Transactions on Computers*, the *IEEE Transactions on SMC-A*, and the *International Journal of Computers and Applications*, as an associate editor. He is a senior member of the IEEE and the IEEE Computer Society.