SECURITY-AWARE SCHEDULING FOR REAL-TIME SYSTEMS

by

Tao Xie

A Dissertation

Presented to the Faculty of

The Department of Computer Science at the New Mexico Institute of

Mining and Technology

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy in Computer Science

Under the Supervision of Professor Xiao Qin

Socorro, New Mexico

May, 2006

ABSTRACT

Over the last decade, clusters have become the fastest growing platforms in high-performance computing. More recently, Grids were emerging as next generation computing platforms for large-scale computation and data intensive problems in industry, academic, and government organizations. Meanwhile, an increasing number of real-time applications running on clusters and Grids have mandatory security requirements in addition to stringent timing constraints. Conventional real-time scheduling algorithms developed for clusters and Grids, however, either disregard applications' security needs, and thus expose the applications to security threats, or run applications at inferior security levels without optimizing security performance. In recognition that many applications running on clusters and Grids demand both real-time performance and security, in this dissertation research we investigate the problem of scheduling real-time applications with various security requirements.

First, we propose a security middleware model (or SMW for short) from which security-sensitive real-time applications are enabled to exploit a variety of security services to enhance trustworthy executions of the applications. A quality of security control manager (QSCM), a centrepiece of the SMW model, has been designed to achieve a flexible trade-off between overheads caused by security services and system performance. Next, we build a security overhead model that can be used to reasonably

measure security overheads experienced by the security-sensitive applications. In light of the security overhead model, an array of security-aware real-time scheduling schemes have been developed by incorporating existing real-time scheduling policies into our novel security-aware heuristics. These security-aware scheduling schemes, which play an important role in the QSCM module, are capable of maximizing quality of security for real-time applications running on clusters and Grids. Comprehensive experimental results based on synthetic traces, real-world traces, and real-world applications show that compared with existing scheduling algorithms our security-aware scheduling schemes significantly improve security of clusters and computational grids while achieving consistently high levels of schedulability.

To my parents

and my wife Zhixiang Wang

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to my advisor, Dr. Xiao Qin, for his supervision, support, patience, and encouragement throughout my Ph.D. study. His technical advice was essential to the completion of this dissertation and has taught me innumerable lessons and insights on the workings of academic research in general and security-aware real-time systems in particular. During the course of this dissertation research, Dr. Qin is the very person with whom I have discussed all my approaches to the problem of security-aware scheduling. It is really my fortune to be supervised by Dr. Qin who has been self-motivated and hard-working with his graduate students. More importantly, I have benefited a lot from his student-oriented spirit, which in turn encourages me to become an assistant professor to be loved by my students.

I would also like to thank Dr. Andrew Sung for his continuous support since the first day I joined New Mexico Tech. As an advisor for my M.S. study at New Mexico Tech, he supervised me in information security area and supported me as a Research Assistant in summers. Being the chair of the Computer Science Department, he has been making the best effort to provide graduate students like me with financial aid and a comfortable working environment. Without his endless support, this dissertation research at New Mexico Tech would have never been successful.

I am grateful to Dr. Lorie Liebrock not only for reviewing my dissertation, but

more importantly for showing me a way of efficient teaching. I have learned a lot from Dr. Liebrock during the two semesters when I was her Teaching Assistant for the Computer Architecture course. In addition, her insightful feedback helped me further improve the presentation of the dissertation in many ways.

I am indebted to Dr. Aly I. El-Osery for reviewing my dissertation. Additionally, he wrote me a reference letter, which enhances my chances in my future career. I would like to express my gratitude to Dr. Dongwan Shin for writing me a recommendation letter. I am equally grateful to him for discussing with me the security overhead model, a critical part of my dissertation.

I thank Dr. Hamdy Soliman, Dr. Subhasish Mazumdar, Dr. Horst Clausen and Dr. Jean-Louis Lassez for their help during the course of my Ph.D. study in the Department of Computer Science at New Mexico Tech. I would like to thank Miss Francesca Denton for helping me a lot on preparing my reference letters.

I wish to thank all members of our research group including Mais Nijim, Mohammed Alghamdi, and all my other friends.

I owe a debt of gratitude to Jianxin Wang for teaching me how to drive using his own car. Also, I want to thank Yuqi Du, one of my best friends, for his great help.

I thank my father Chengdi Xie and mother Qingru Wu for their unlimited support and strength. Without their dedication, I could not have pursued a Ph.D. degree in Computer Science at the New Mexico Institute of Mining and Technology.

Last and most importantly, I am grateful to my wife Zhixiang Wang, whose endless encouragement coupled with love has been my source of inspiration. I would have never been successful without her input.

This dissertation is accepted on behalf of the faculty
of the Institute by the following committee:

Academic Adviser

Research Advisor

Committee Member

Committee Member

Committee Member

4/3/2006

Date

I release this document to New Mexico Institute of Mining and Technology

4/4/2006

Students Signature Date

TABLE OF CONTENTS

Chapter

LIST OF TABLES

LIST OF FIGURES

Chapter 1

1 Introduction

In the past decade, a growing number of real-time applications have been developed and deployed both in clusters [5][39][64][65] and Grids [29]. A cluster is a parallel processing system comprising a group of interconnected commodity computers, which work together as an integrated computing platform [65]. Clusters have become the most cost-effective computational platforms for scientific applications [61][62]. As typical scientific simulation and computation require a large amount of compute power, it is appealing to apply clusters where computational nodes are connected through high-speed networks to meet needs of complex scientific computing [5][66]. A computational Grid (or Grid for short) is a collection of geographically dispersed computing resources with distributed control, providing a large virtual computing system to users. With rapid advances in processing power, network bandwidth, and storage capacity, Grids are emerging as next generation computing platforms for large-scale computation and data intensive problems in industry, academic, and government organizations. It is worth

noting that security is of critical importance for a wide range of real-time applications [5][6][7][22][60][75]. However, conventional wisdom on design of real time systems is inadequate for security-sensitive real-time applications due to the lack of consideration for the application's security needs.

Since scheduling algorithms play a key role in obtaining high performance in cluster computing [85][106] and Grid computing environments [18][79], we believe that a feasible way of accommodating security-sensitive applications in a real-time system is to design security-aware scheduling schemes. In particular, security-aware scheduling algorithms can improve security of real-time applications while maintaining a high level of performance for clusters and Grids. This dissertation work investigated security-aware scheduling mechanisms and policies for real-time applications with security requirements running on clusters and Grids.

This chapter first manifests the problem statement in Section 1.1. In Section 1.2, we describe the scope of this research work. Section 1.3 summarizes the main contributions of the dissertation, and Section 1.4 presents the outline of the dissertation.

1.1 Problem Statement

In this section, we start with an overview of security demands of real-time systems running on clusters and Grids. Section 1.1.2 describes a real-time stock quote update and trading system [26] to illustrate characteristics of existing security-critical real-time systems. Finally, Section 1.1.3 presents motivation for the dissertation research by highlighting the lack of solution for scheduling applications with timing and security constraints.

1.1.1 Why Security Is Needed?

An increasing number of real-time applications have security constraints because sensitive data and processing require special safeguards against unauthorized access [15] [27]. For example, a variety of military real-time applications such as aircraft control systems [2] running on parallel and distributed systems like clusters require security protections to completely fulfill their security needs.

Since clusters and Grids are built to execute a broad spectrum of unverified user-implemented applications from a vast number of different users, both applications and users can be sources of security threats [104]. For instance, the vulnerabilities of applications can be exploited by hackers to compromise clusters and Grids and malicious users can access systems to launch denial of service attacks. Even a legitimate user may tamper with shared data or excessively consume computing cycles to disrupt services available to other cluster users [104]. Even worse, many existing cluster and Grids computing environments have not employed any security mechanism to counter security threats [21][22]. As such, it is mandatory to deploy security services to guard security-critical applications running on clusters and Grids. Snooping, alteration, and spoofing are three common attacks in cluster environments and, therefore, we considered three security services (authentication service, integrity service, and confidentiality service) to guard against the common threats. Snooping, an unauthorized interception of information can be countered by confidentiality services. Alteration, an unauthorized change of information can be countered by integrity services. Spoofing, an impersonation of one entity by another, can be countered by authentication services [12][9]. With these three security services in place, users can flexibly select the security services to form an

integrated security protection against a diversity of threats and attacks in a cluster or Grid computing environment.

1.1.2 An Example of Security-Sensitive Real-Time Applications

Figure 1.1 illustrates a real-time stock quote update and trading system with high security demands. In this system [26], requests submitted from a business partner and responses from an enterprise's back-end application (the terms request and task are used interchangeably throughout this section) have deadlines and security requirements. Both timing and security requirements must be met by a server located between the business partners and enterprise back-end applications. In this case, the server performs security operations on behalf of all its clients. Each task submitted by a large group of clients explicitly specifies a range of security levels in addition to the deadline before which the task must be completed. The server facilitates required security mechanisms on top of a request or a response in an out-of-the-box integration manner. In general the server performs the following security operations on behalf of its clients [26]:

- Establishes secure connections with business partners and back-end applications
- Applies and verifies digital signatures
- Authorizes access based on digital certificates
- Validates credentials in real time using public key infrastructure
- Encrypts and decrypts requests and responses

Furthermore, the server can judiciously select a suitable security level from the range of security service levels, which are predefined combinations of transport and message security mechanisms. Typical security levels in a real-time quote and trading system are listed as follow [26]:

- Routing only
- Routing + message security
- Routing + SSL
- Routing + SSL + message security
- Routing + SSL + client authentication
- Routing + SSL + message security + client authentication

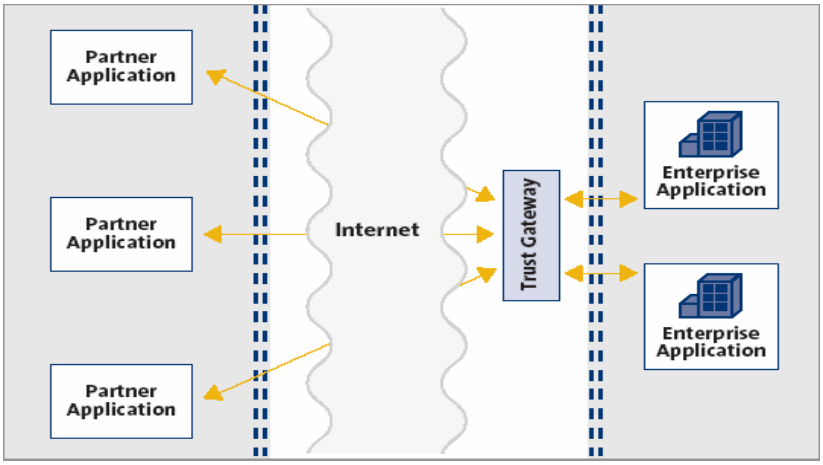


Figure 1.1 A real-time stock quote update and trading system

Different security levels, of course, impose various security overheads with respect to CPU and memory usage. While a real-time system may automatically provide high quality of security for some tasks by increasing security levels at the cost of high overheads, the system can intentionally reduce the quality of security for other tasks in order to improve guarantee ratio measured as a fraction of total submitted tasks that can be completed before their deadlines.

1.2 Motivations

Security-critical real-time applications such as stock quote update and trading systems and military aircraft flight control systems have mandatory security requirements in addition to stringent timing constraints. Existing real-time scheduling algorithms, however, either disregard applications' security needs and thus expose the applications to security threats, or run applications at inferior security levels without optimizing security performance. In recognition that many applications running on clusters or/and Grids demand both real-time performance and security, we investigate in this dissertation the problem of scheduling real-time applications with various security requirements.

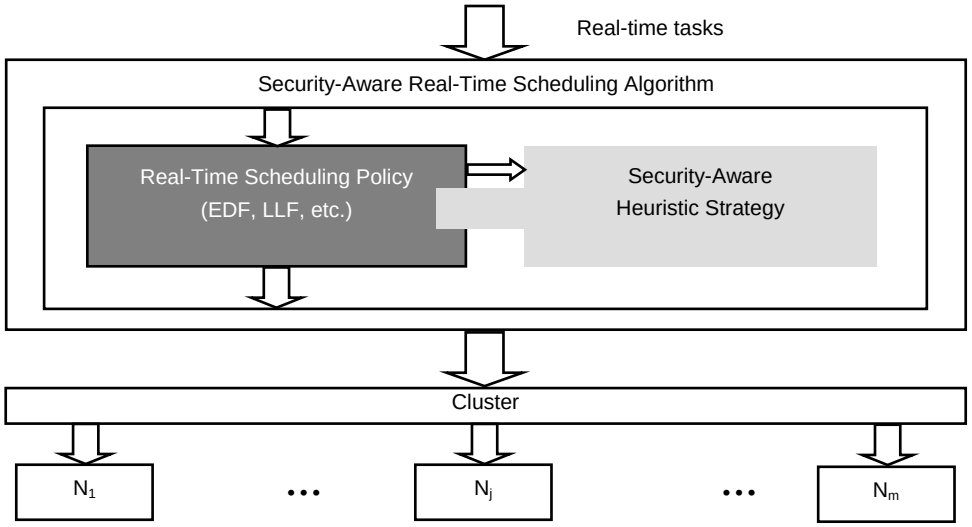


Figure 1.2 Security-aware real-time scheduling framework

To address timing and security constraints of real-time applications, we proposed an array of security-aware heuristic strategies, which integrate security requirements into scheduling for real-time applications on clusters or Grids. Figure 1.2 depicts the relationship between real-time scheduling algorithm and our security-aware heuristic strategy. The scheduling core implements logic and timing mechanisms for waiting, and

relies on the security-aware heuristic strategy to decide quality of security for newly arrived tasks. The security-aware heuristic strategy is independent of scheduling policies, and it is implemented as a module that works in concert with real-time scheduling policies. Therefore, it is easy to integrate the security-aware heuristic strategy into any real-time scheduling policy.

1.3 Scope of the Research

The dissertation research focuses on scheduling real-time applications with security requirements running on clusters and Grids.

We assume that the underlying security services such as confidentiality and integrity are deployed in clusters and Grids. For sake of simplicity, we ignore the overhead caused by key management, because the key management overhead can be included as part of security overhead. It is assumed that all tasks considered in this research are non-preemptive. For security services, we only considered three common ones, namely confidentiality, integrity, and authentication.

In this dissertation study, we have explored the following. First, we have proposed a security-aware system framework, from which security-sensitive real-time applications are able to exploit a variety of security services to enhance trustworthy executions of the applications. Second, we have built a security overhead model that can be used to reasonably measure security overheads incurred by security-critical applications. Third, we have developed a novel security-aware scheduling strategy for a set of independent real-time tasks. Fourth, we have implemented two security-aware strategies for parallel real-time applications, where precedence constraints among different tasks need to be taken into account. Fifth, we have extended our security-aware scheduling schemes to

heterogeneous clusters. Last but not least, we have devised an effective scheduling scheme for Grid computing environments.

1.4 Contributions

Improving quality of security is increasingly becoming an important issue in the design of real-time systems, which are indispensable for conducting business in government, industry, and academic organizations. This dissertation research addresses the issue of maximizing quality of security for real-time systems through scheduling. In what follows, we summarize the major contributions of the dissertation study.

- **A Security Middleware Model for Real-time Applications:** We proposed a security middleware (SMW) model from which security-sensitive real-time applications are enabled to exploit a variety of security services to enhance trustworthy executions of the applications.
- **A Security Overhead Model:** We proposed an effective model that can approximately, yet reasonably, measure security overheads experienced by tasks with security requirements. In light of the security overhead model, schedulers are able to incorporate security overheads into the process of scheduling tasks.
- **Security-Aware Scheduling for Homogeneous Clusters:** We presented a novel security-aware heuristic strategy (SAREC) for real-time applications on homogeneous clusters. This strategy paves the way to the design of security-aware real-time scheduling algorithms.
- **Consideration of the Heterogeneity of Resources:** Many existing clusters are heterogeneous in terms of resources including but not limited to CPU, memory, and

disk storage. Since heterogeneity in resources inevitably complicates the scheduling issue of real-time applications, we devised a security and heterogeneity driven scheduling algorithm to improve security of real-time applications on heterogeneous clusters.

- **Supporting for Parallel Applications:** We extended our work in security-aware scheduling for sequential jobs by developing a security-driven task allocation scheme for parallel applications with deadline, security, and task precedence constraints.
- **Improving Security for Grids:** In addition to our research in scheduling issues related to clusters, we designed and developed a dynamic real-time scheduling algorithm, which is capable of enhancing quality of security for real-time applications running on Grids.

1.5 Outline

This dissertation is organized as follows. In Chapter 2, a brief survey of related work is presented.

In Chapter 3, we propose a security middleware model (or SMW for short) from which security-sensitive real-time applications are enabled to exploit a variety of security services to enhance trustworthy executions of the applications.

To measure security overhead incurred by security requirements of real-time applications, we build in Chapter 4 a security overhead model.

In Chapter 5, we study the problem of scheduling a set of independent real-time tasks with various security requirements on homogeneous clusters.

In Chapter 6, a new security-conscious scheduling scheme for parallel real-time applications is presented and its performance is evaluated.

In Chapter 7, we present a concept of security heterogeneity and a new security- and heterogeneity-driven scheduling algorithm, which strives to maximize the probability that parallel applications are executed on time without risk of being attacked.

In Chapter 8, we outline a dynamic real-time scheduling algorithm, which aims at enhancing quality of security for real-time applications running on Grids.

Finally, Chapter 9 summarizes the main contributions of this dissertation and comments on future directions for this research.

Chapter 2

2 Related Work

Clusters have become popular as high-performance computing platforms for a variety of applications, and Grid is a promising next generation high-performance computing platform. This chapter briefly discusses existing techniques found in the literature that are most relevant to this dissertation research from two perspectives: real-time scheduling on clusters and Grids, and security needs for real-time applications running on clusters and Grids.

2.1 Real-Time Scheduling on Clusters and Grids

The problem of real-time scheduling has been extensively studied in the past both theoretically and experimentally. Real-time scheduling algorithms generally fall into two categories: static (off-line) [1] and dynamic (on-line) [19][44]. Many scheduling algorithms assume that real-time tasks are independent with one other [83], whereas others can schedule tasks with precedence constraints [1]. Conventional real-time scheduling algorithms like Rate Monotonic (*RM*) algorithm [51], Earliest Deadline First

(EDF) [77], and Spring scheduling algorithm [67] were successfully applied in real-time systems. This section presents a summary of work related to real-time scheduling techniques applied in clusters and Grids. Specifically, we describe the important features in a wide range of real-time scheduling approaches. These features include static and dynamic scheduling (See Section 2.1.1), considerations for system heterogeneity (See Section 2.1.2), and parallel jobs versus sequential jobs (See Section 2.1.3).

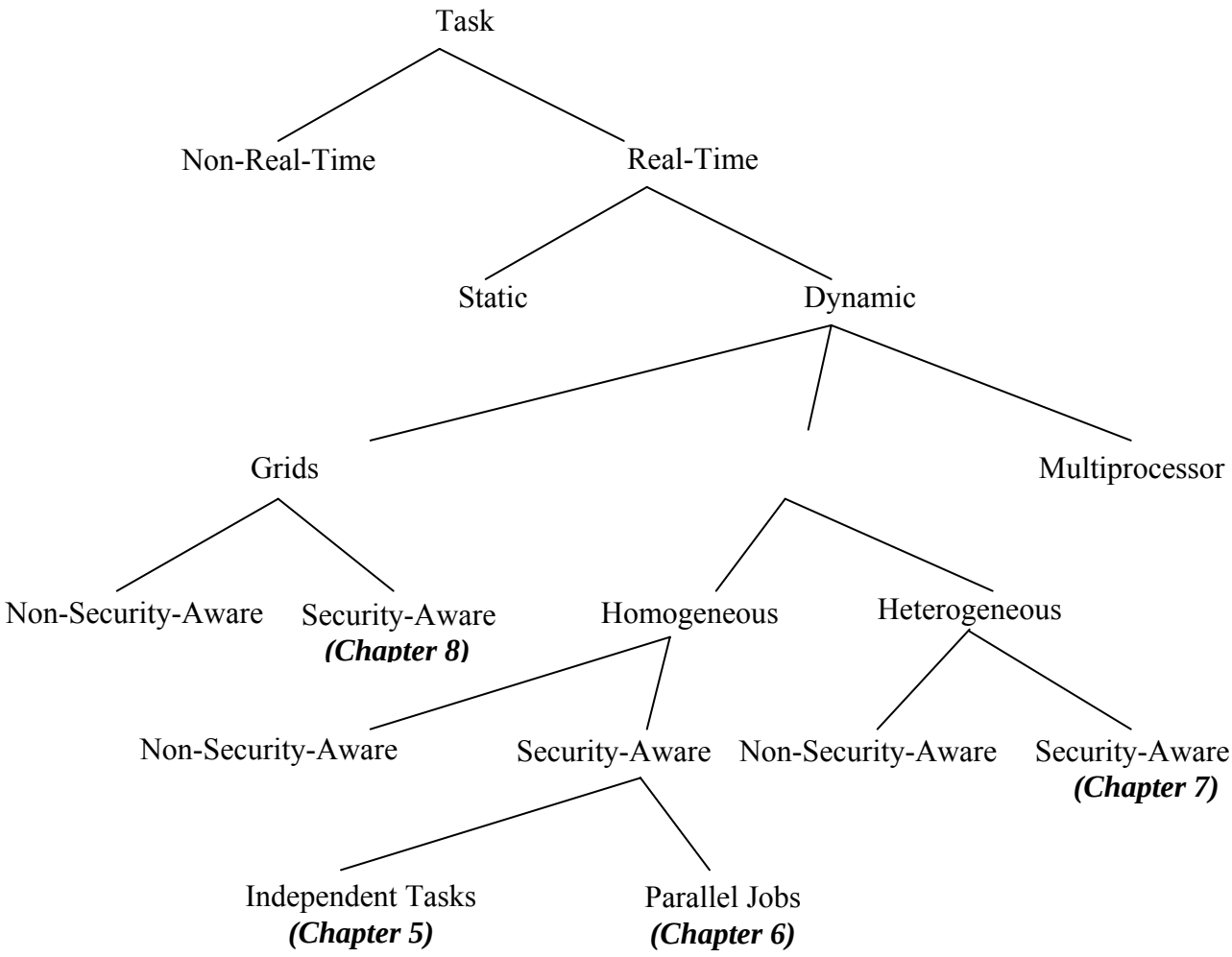


Figure 2.1 A simplified taxonomy of the approaches to scheduling

2.1.1 Static vs. Dynamic Scheduling: A Simplified Taxonomy

We can classify existing scheduling approaches into two broad categories: static and

dynamic (See Figure 2.1). Each category can be further divided into some subcategories with certain specific attributes. These attributes include control philosophy (centralized vs. decentralized), resource targeted for optimization (CPU time or energy consumption), type of jobs (sequential vs. parallel, real-time vs. non-real-time, etc.), nature of underlying system (homogeneous vs. heterogeneous), etc. Figure 2.1 shows a simplified taxonomy of scheduling schemes most relevant to this dissertation research.

In static scheduling, the characteristics of a job, such as its task execution times, task dependencies, task communications and synchronization are known a priori [21][31]. Therefore, scheduling can be done off-line during compile-time. A large body of work has been done for static scheduling schemes that do not rely on current states of nodes [9][14][21][31][45][48][46]. Since discovering an optimal schedule for a parallel application on a multiprocessor system is an NP-complete problem in general, researchers proposed numerous heuristics [48]. Kwok and Ahmad proposed a taxonomy that classifies these algorithms into different categories and compared them in terms of performance and time-complexity [48]. Boeres and Rebello investigated the problem of static task scheduling under the LogP model and presented both theoretical and experimental results for a cluster-based task duplication methodology [14]. Barbosa *et al.* proposed a list scheduling algorithm to minimize total schedule lengths of parallel jobs represented by DAGs [9].

On the other hand, dynamic scheduling in the absence of a priori information is done on-the-fly based on system states [3][59]. Hamidzadeh and Lilja examined a class of self-adjusting dynamic scheduling (SADS) algorithms that centralizes assignments of tasks to processors [36]. Their experimental results demonstrated that the centralized scheduling

outperforms self-scheduling algorithms even when only a small number of processors are available. Kalogeraki *et al.* presented a dynamic scheduling algorithm that examines the computation times, real times and resource requirements of application tasks to determine a feasible schedule for the method invocations [45]. Yu and Bhattacharya proposed real-time traffic management algorithms over a multi-hop optical network [102]. The objective of their work is to schedule hard real-time messages as many as possible following a priority ordering. Recently, some dynamic scheduling advances were accomplished in the context of Grids. Viswanathan *et al.* proposed a resource conscious dynamic scheduling strategy to handle large volume computationally intensive loads in a grid system involving multiple sources and sinks/processing nodes [87]. Wang *et al.* proposed an adaptive and dynamic scheduling method, called most fit task first (MFTF), for a class of computational grids, which are characterized by heterogeneous computing nodes and dynamic task arrivals [89].

2.1.2 Homogeneous vs. Heterogeneous Scheduling

From another perspective, scheduling approaches can be classified into two camps: homogenous and heterogeneous (See Figure 2.1). The focus of homogeneous scheduling schemes is to improve performance of homogeneous clusters. For example, Hagras and Janecek proposed a low complexity algorithm based on list-scheduling and task-duplication on a bounded number of fully connected homogeneous machines. The algorithm is called critical unlisted parents with fast duplicator (CUPFD), which consists of two phases: the listing phase that is based on list-scheduling, and a low complexity machine assigning phase based on task-duplication [34]. Karageorgos and Karatza examined the efficiency of two task routing strategies-one static and one adaptive-and

three non preemptive task scheduling policies in conjunction with job resequencing before departure [46]. Burchard *et al.* presented new schedulability conditions for homogeneous multiprocessor systems where individual processors execute the rate-monotonic scheduling algorithm. The conditions are used to develop new strategies for assigning real-time tasks to processors. The performance of the new strategies is shown to be significantly better than those presented in the literature [17]. Note that our algorithms presented in Chapters 5 and 6 are scheduling schemes for homogeneous clusters.

On the other hand, heterogeneous scheduling approaches attempt to boost the performance of heterogeneous clusters, which comprise a variety of processing nodes with different performance capacities in computing power, memory capacity, and disk speed. Berten *et al.* studied the distribution of sequential jobs and system behaviours in heterogeneous computational grid environments where brokering is done in such a way that each computing element has a probability to be chosen proportional to its number of CPUs and its relative speed [11]. A probabilistic model of a client/server multimedia heterogeneous system based on the theory of Markov processes was proposed by Santos *et al.* Their model also provides answers to some fundamental engineering design questions related to disk price/performance tradeoffs [70]. Bajaj and Agrawal introduced a Task duplication-based scheduling Algorithm for Network of Heterogeneous systems (TANH), which provided optimal results for applications represented by Directed Acyclic Graphs (DAGs) [8]. A heuristic dynamic scheduling scheme for parallel real-time jobs executing on a heterogeneous cluster was presented by Qin and Jiang [65]. In [80], a mathematical framework is presented that models the matching of subtasks to

machines, scheduling of subtasks' computation, scheduling of communication steps, and selection of sources of shared data items. The goal of this study is to generate a provably optimal scheme for communicating shared data among subtasks as an enhancement to any given matching and scheduling.

In the past decade, the scheduling techniques for heterogeneous systems have been extensively investigated. However, security-aware scheduling in the context of heterogeneous systems is an interesting area yet to be explored. Therefore, in Chapter 7 we examine the problem of scheduling for parallel applications executing on heterogeneous clusters.

2.2 Security Issues in Clusters and Grids

Nowadays security is of critical importance for a wide range of real-time applications. Section 2.2.1 describes security issues in the context of clusters and Grids. Section 2.2.2 summarizes previous work related to QoS (Quality of Service), where security is one of parameters of QoS.

2.2.1 Security Needs for Real-Time Applications

Increasing attention has been directed toward the issue of security in the context of clusters, because efficient and flexible security has become a baseline requirement. Apvrille and Pourzandi developed a new security policy language named distributed security policy, or DSP, for clusters [6]. Wright *et al.* proposed a security architecture for a network of computers bound together by an overlying framework that can be used to provide users a powerful virtual heterogeneous machine [95]. The language offers a precise way to customize security of clusters. Yurcik *et al.* developed tools for managing

cluster security via process monitoring [103]. Connelly and Chien proposed an approach to protecting tightly-coupled, high-performance component communications [22]. Azzedin and Maheswaran applied the notion of "trust" into resource management of a large-scale wide-area system [7]. However, the security techniques mentioned above are not appropriate for real-time applications due to the lack of ability to express and handle timing constraints. In addition, security has been become one of the biggest concerns in Grids. Humphrey *et al.* examined the state of the art in securing a group of activities and introduced new technologies that promise to meet security requirements of Grids [42].

Some work was done to incorporate security into a variety of real-time applications. George and Haritsa proposed concurrency control protocols to support applications with real-time and security requirements [32]. Ahmed and Vrbsky developed a secure optimistic concurrency control protocol that can make trade-offs between security and real-time requirements [4]. Son *et al.* proposed a way of trading off quality of security to achieve required real-time performance [74]. In [75], a new scheme was developed to improve timeliness by allowing partial violations of security. Our work is fundamentally different from the above approaches because they are focused on concurrency control protocols whereas the goal of this dissertation research is to develop security-aware real-time scheduling algorithms.

However, most existing security techniques are not appropriate for real-time applications due to the lack of ability to express and handle timing constraints. Xie *et al.* proposed an array of security-aware scheduling algorithms for single machines [101], clusters [100] and grids [99]. Very recently, Song *et al.* proposed security-driven scheduling algorithms for grids [76]. This study is by far the closest one to our algorithm

(see Chapter 7). The main difference between our study presented in Chapter 7 and theirs are five-fold. (1) Their algorithms can efficiently support non-real-time applications, whereas our algorithm is designed for parallel applications with timing constraints. (2) Their study was focused on grids instead of clusters. (3) It was assumed in their algorithms each grid site offers one conceptual security level. However, our algorithm factors in practical security services with multiple security levels. (4) Our algorithm takes into account of heterogeneities in security and computation, whereas their algorithms support homogeneous computing resources. (5) Their algorithms made use of a failure model that did not take execution times into consideration. 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.

2.2.2 Quality of Security and Security Overhead

Since security is one dimension of QoS (Quality of Security), we briefly review QoS-aware middleware, which has been extensively studied in the past [2][38][55]. Huang *et al.* proposed a middleware-oriented Global Resource Management System, or GRMS, which provides distributed applications with end-to-end QoS negotiation and adaptation [41]. Nahrstedt et al. designed a QoS-aware middleware that can offer a new generation QoS-sensitive applications such as media streaming and e-commerce with QoS support [55]. However, these two middleware systems are non real-time in nature, meaning that they are inadequate for parallel and distributed real-time systems. Abdelzaher *et al.* presented a scheme for QoS negotiation in real-time applications [2]. This approach provides a generic way to express application-level semantics to control how application QoS is to be degraded under overload or failure conditions [2]. In addition, Abdelzaher *et*

al. demonstrated that their method enables QoS gracefully degradation under conditions in which traditional schedulability analysis and admission control schemes fail. Although the above works addressed applications' QoS requirements in parallel and distributed systems, none of them paid attention to real-time applications' security requirements, which are increasingly becoming critical in real-time systems. This dissertation research is orthogonal and complementary to the above approaches in the sense that the security middleware model centered around security services is focused on security needs of real-time applications.

The security middleware model (SMW, see Chapter 3) provides a way of explicitly specifying security requirements of real-time applications running on a parallel and distributed computing platform. It is indispensable for the model to be aware of extra resource overhead incurred by security requirements of applications because the model has to achieve an optimized trade-off between system security and performance. To the best of our knowledge, the way of calculating costs of security service has received little attention. Irvine *et al.* proposed a model of computing costs for quality of security service [43]. In their approach application's security requirements are specified by a security vector, which is composed of an array of sub-vectors with each sub-vector being a particular security service used [43]. Each sub-vector could consist of two components: a security service name and a security level range of the security service. Wang *et al.* presented a security measurement framework, which is based on theory and practice of formal measurements [90]. Their work provided us an insightful view of a future direction of security measurement. In our previous work [101], we proposed a practical security overhead model to estimate the CPU time overhead of some commonly used

security services. Our security overhead model leveraged the results in [98][99], which provided the CPU time cost for primitive security operations such as encryption and integrity check. Take encryption operation for example, Nahum *et al.* in [56] offers the performance of ten widely used encryption algorithms in terms of mega bytes per second (MB/s) on a 175 MHz Dec Alpha600 machine. 3DES, the strongest yet slowest encryption algorithm among the alternatives, can encrypt 6.25 MB data per second on the computer. The computational overhead caused by security operations mainly depends on cryptographic algorithms and size of data to be protected. Detailed information regarding a way of quantitatively measuring security overhead can be found in our previous work [98][99].

Chapter 3

3 A Security-Aware Middleware Model

Recently, real-time applications with security requirements increasingly emerged in clusters and large-scale distributed systems like Grids. However, complexities and specialties of diverse security mechanisms dissuade users from employing existing security services for their applications. To effectively tackle this problem, in this chapter we propose a security middleware model or SMW from which security-sensitive real-time applications are enabled to exploit a variety of security services to enhance trustworthy executions of the applications. A quality of security control manager (QSCM), which is a centrepiece of the SMW model, is designed and implemented to achieve a flexible trade-off between overheads caused by security services and system performance, especially under situations where available resources are dynamically changing and insufficient. A security-aware scheduling mechanism, which plays an important role in QSCM, is capable of maximizing quality of security for real-time applications running on clusters and Grids.

The rest of the chapter is organized as follows. Section 3.1 introduces the architecture of our SMW model. Section 3.2 concludes the chapter with some comments on future work.

3.1 Security Middleware Model (SMW)

Middleware is software that sits between two or more types of software and translates information between them. It is used to solve computer clients' heterogeneity and distribution issues by offering distributed system services that have standard programming interface and protocols [10]. We refer to these system services as middleware services, because they reside in a layer between networking, operating system software and specific applications. In this section we propose a security middleware (SMW) model, which aims at meeting security requirements of a variety of applications and improving performance of distributed real-time systems. Section 3.1.1 presents an overview of the architecture for the SMW model. Detailed functional descriptions of each component of the SMW model can be found in Section 3.1.2. Section 3.1.3 illustrates two approaches to specifying applications' security requirements. Section 3.1.4 provides an analytical model for the local schedulability analyzer.

3.1.1 Architecture of the SMW Model

The SMW model consists of a user interface, a framework, low-level security service APIs, a quality of security control manager, and security middleware services (see Figure 3.1).

The SMW model provides two different types of user interfaces, namely, a professional user interface and a normal user interface. The professional user interface is

an interface between developers (e.g., programmers) and applications being developed. An editor, a compiler and a debugger are essential components of the professional user interface. Programmers are allowed to directly access the low-level security service APIs, thereby efficiently constructing applications with various security functions. A normal user interface sits between a normal user and the framework. By using the normal user interface, usually an IDE (integrated development environment), a normal user such as a system administrator can leverage the framework to readily create applications with security requirements.

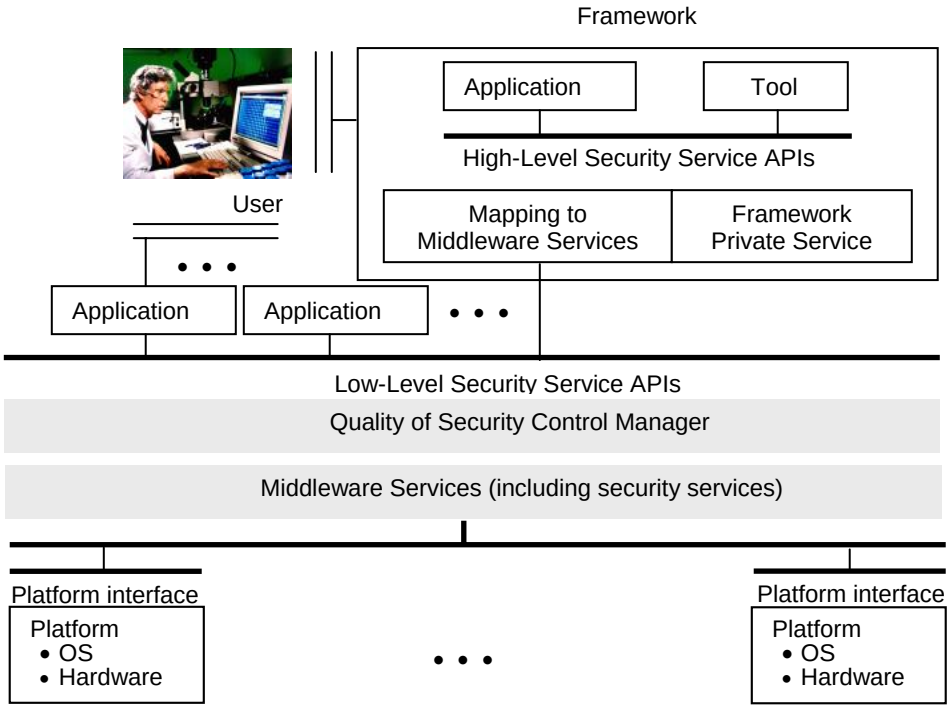


Figure 3.1 Security middleware architecture

A *framework* is a software environment that is designed to simplify application development and system management for a specialized application domain [10]. The framework illustrated in Figure 3.1 is composed of a set of high-level security service APIs, an array of tools, a security middleware-service mapping module, and framework-

private middleware services. The functionality of the framework is two-fold. First, it provides developers an efficient computing environment in which security-aware applications can be rapidly developed. Second, the framework makes it possible for users to manipulate security-related system parameters. As a result, there is no need for developers and users to directly access low-level security service APIs, which are, in most cases, complicated to use. The high-level security service APIs may be (1) an abstraction of low-level security service APIs for the underlying security middleware services, or (2) a new set of APIs that encapsulate the low-level security service APIs. When the high-level APIs are different from their low-level peers, they may add value by specializing the user interface, simplifying the low-level APIs, or import framework-private middleware services. The applications within the framework are administration applications from which the users (including administrators and programmers) can manage and configure multiple security services by employing the high-level APIs with the assistance of some tools. The objective of the tools in the framework is to simplify the use of the high-level APIs. For example, a security service virtualization tool offers users a visible table that demonstrates all currently available security services and their corresponding costs (see Table 3.1 in Section 3.1.3). The security middleware services mapping module is responsible for translating the high-level security service APIs into their corresponding low-level counterparts. Framework-private services provide specific functions in addition to the underlying middleware services to meet the framework's own needs.

The low-level security service APIs are programming interfaces through which underlying security services included in the middleware services can be invoked. We can

implement our low-level security service APIs based on the Generic Security Service API described in [94], which allows a calling application to authenticate principle identity associated with a peer application, to delegate rights to a peer application, and to exploit security services such as confidentiality and integrity on a per-message basis [94]. A sample API routine could be *gss_verify_mic()*, which can check a message integrity code (MIC) against a message to verify integrity of a received message. Another example routine is *gss_indicate_mechs()* that determines available underlying authentication mechanisms.

Quality of security control manager (QSCM) is a module needed for optimizing applications' security requirements based on available system resources. Conceptually, it is an engine for security-critical real-time systems to achieve a high system performance in terms of quality of security and schedulibility. Detailed description of QSCM will be given in Section 3.1.2. A middleware service is a generic service that operates between platforms and applications (see Figure 3.1). The middleware service, which is defined by APIs and supported protocols [10], has several features that differ itself from general-purpose applications or platform-oriented services. Specifically, the middleware service is distributed, capable of running on multiple platforms, and supporting standard interfaces and protocols. Among the middleware services, authentication service, auditing service, encryption service and access controller are commonly used security services in a distributed system. For instance, authentication service provides functions to an application related to establishing, verifying, and transferring a person or a process. These security middleware services furnish a set of standard APIs (e.g., low-level security service APIs), which can be invoked in applications. The services are standard

routines, which can be implemented using programming languages such as C and Java. For example, a Java Security Service Module is a commercial product that facilitates the above services implemented as classes.

3.1.2 Quality of Security Control Manager (QSCM)

QSCM (Figure 3.2) is a centerpiece of the SMW model because it can optimize the quality of security services requested by applications while maintaining high system performance. The input of the QSCM module is a security service attribute-value vector specified by users, and the output is an array of selective values for each required security service. The most important abstraction in our QSCM module is *security level*, which is used to indicate the strength or safety degree of a particular security service.

A security service is implemented by a particular security mechanism. For example, encryption, a security mechanism, provides a means to implementing confidentiality, which is a security service. Thus, the strength of a security service is mainly decided by the robustness of the security mechanism that implemented it. Further, the strength of the security mechanism largely depends on (1) how rigorously the security algorithm is tested, (2) how long it has been used, and (3) how robust it is under attacks performed against it [90]. From a normal user's standpoint, a security level may be a subjective and qualitative value like "low", "medium", and "high". For a security professional, on the other hand, the security level could be a quantitatively measured value such as 0.3, a normalized value when setting the strongest security mechanism as 1. In the latter case, security level is a relatively objective value obtained by some reasonable and practical measurement methods. In addition, security levels are represented in terms of security

parameters whose semantics only need be known to the user and the service provider (e.g., security middleware service).

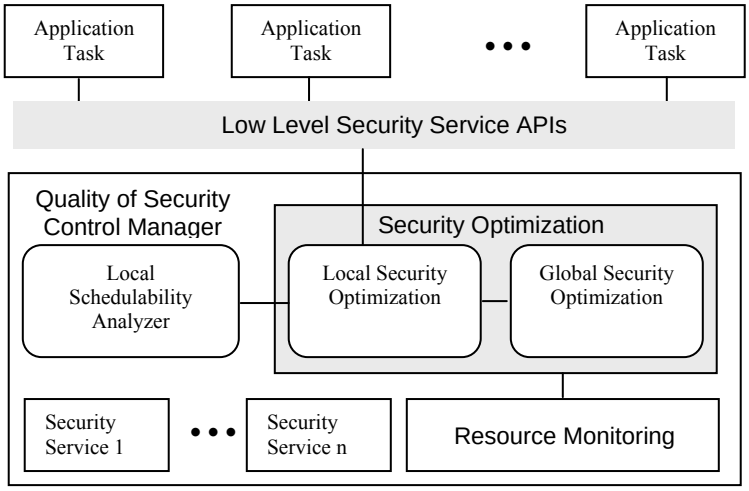


Figure 3.2 Quality of security control manager

Please note that security mechanisms are not independent of one another. Rather, it is common that multiple security mechanisms are needed in order to form an integrated security solution. For example, authentication must be used in concert with message integrity. The SMW model offers users an array of basic security mechanisms so that they can select multiple services to form an integrated security solution. It is users' responsibility to make a meaningful combination of fundamental security mechanisms. The local security optimization module, which will be described shortly in this subsection, can assist users to accomplish this goal.

A *security range,* which is a scope, contains multiple distinct security levels for a particular security service. The lowest value in a security range indicates the minimal security strength mandated by the user, while the highest value implies the maximal security strength necessary for the user and all the values above should not be considered. We will discuss the security level specification in section 3.3.

The QSCM runs on top of middleware services and uses the Resource Monitoring module to monitor the underlying available resources. A user may submit a task T_i along with its security requirements expressed by a vector of security ranges, e.g., $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, where T_i requires q security services. S_i^j is the security range of the jth requested security service.

The security optimization module, which plays a key role in QSCM, is responsible for choosing the most appropriate point s_i in space S_i, e.g., $s_i = (s_i^1, s_i^2, \ldots, s_i^q)$, where $s_i^j \in S_i^j$, $1 \le j \le q$. The objective of the security level selection is to maximize overall utility in terms of quality of security (see Section 3.4).

The local schedulability analyzer aims at checking whether or not the selected security levels can be supported under current workload conditions. With the assistance of the local schedulability analyzer, the security optimization module performs admission control on arrival application tasks.

The scheduling mechanism has to make use of the schedulability analyzer and the security optimization module to measure the security benefits gained by each admitted task. In particular, the security benefit of task T_i is quantitatively modeled as the following security level function.

$$SL(s_i) = \sum_{j=1}^{q} w_i^j s_i^j \text{, where } 0 \le w_i^j \le 1 \text{ and } \sum_{j=1}^{q} w_i^j = 1, \qquad (3.1)$$

where w_i^j is the weight of the jth security service. Note that it is programmers' responsibility to define the weights to reflect relative priorities given to the required security services.

Suppose X_i is all possible schedules for task T_i generated by the scheduling mechanism, and $x_i \in X_i$ is a scheduling decision. The schedulability analyzer considers x_i a feasible schedule if (1) the security requirements are satisfied, and (2) its deadline can be met. Given a real-time task T_i, the security benefit of T_i is expected to be maximized by the security level controller (See Figure 3.1) under the timing constraint:

$$SB(X_i) = \max_{x_i \in X_i} \left\{ \sum_{j=1}^{q} w_i^j s_i^j(x_i) \right\}, \tag{3.2}$$

where $\min(S_i^j) \le s_i^j(x_i) \le \max(S_i^j)$. $\min(S_i^j)$ and $\max(S_i^j)$ are the minimum and maximum security requirements. The QSCM is focused on maximizing quality of security, which is defined by the sum of the security levels of admitted tasks. More formally, the following security function needs to be maximized, subject to certain timing and security constraints:

$$SV(X) = \max_{X_i \in X} \left\{ \sum_{i=1}^{p} y_i SB(X_i) \right\}, \tag{3.3}$$

where p is the number of submitted tasks, y_i is set to 1 if task T_i is accepted, and is set to 0 otherwise.

The local security optimization module is used to select security levels only for local clients based on local machine resources, while the global security optimization module is launched using a load-sharing algorithm, which can exploit distributed system resources when local resources are insufficient to sustain a client's service requests. In addition, the local security optimization module validates security mechanism selections made by local clients before selecting security levels for them. If a client selects a number of security mechanisms that cannot form a meaningful integrated protection solution, the

local security optimization module will provide a warning message to the client. This function enforces that only practical security solution requests can be granted.

3.1.3 Security Service Requirements Specification

In this subsection we present two approaches to specifying users' security service requirements. One is for professional users, whereas the other is for system administrators. Irvine *et al.* proposed the notion of security range that consists of a set of security levels [43]. Users can define their security requirements for a security service by specifying a security range. To accomplish this goal, our SMW model provides users with a *task submission description language* (TSDL), a vehicle that users can leverage to articulate their security needs for their tasks. Figure 3.3 illustrates an example of the task submission structure (TSS) described in TSDL.

```
DEFINE Task : flight_control
{
Input = (altitude: 1230, heading: 35, …);
Output = (takeoff_distance, climb_rate);
Type = "Real Time";
Period = 80;
Owner = "sqin";
Cmd = "flight_con";
Processor_num= 5;
Data_secured=250;
Priority = 3;
Constraints
• Arch == "INTEL";
• OS == "UNIX";
• Disk >= 480;
• Memory >=128;
• Deadline = 80;
• 0.3 <= Authentication <=0.6;
• 0.4 <= Integrity <= 0.8;
 • 0.5 <= Confidentiality <= 0.9
}
```

Figure 3.3 Task submission structure for flight control

A TSS is a highly flexible and extensible data model that can be utilized to represent multiple security services and constraints in a submitted task. It is a mapping from

attribute names to expressions. For example, Processor_Num is an attribute and the number 5 is its corresponding expression. An expression might be an integer, a string constant, or a combination of complicated expressions constructed with arithmetic and logical operators such as "0.3 <= Integrity <=0.8" (See Figure 3.3).

After a user submits a TSS, the security service constraints will be translated into the high-level security service APIs. To further alleviate users' burden, the framework of the SMW model makes is possible for system administrators to specify security requirement expressions using a higher-level abstraction, e.g., security abstract table.

Table 3.1 illustrates an example of the security abstract table. In this table a user simply needs to point out a qualitative level for each requested security service. Importantly, the user can utilize a visualization tool, e.g., a security function explainer, to glean information pertinent the meaning and cost of each qualitative level. Please note that the values in Table 1 are only for illustration purpose, and the cost should be calculated or estimated using some security cost formulas.

Table 3.1 A Sample Security Abstract Table

Resources / Security Services	CPU (ms)		Memory (kb)		Bandwidth (mbps)	
	Level	Cost	Level	Cost	Level	Cost
Authentication	0.3	13	0.3	0.48	0.3	0.02
Confidentiality	0.2	250	0.2	138	0.2	2.5
Integrity	0.6	197	0.6	73	0.6	5.7

3.1.4 Schedulability Analyzer Modeling

To build the schedulability analyser in QSCM, we propose in this section a model to analytically conduct feasibility check for real-time tasks submitted to a Grid. Let f_i be the

finish time of task T_i. Given a Grid, the finish time of T_i running on it can be expressed as

$$f_i = \begin{cases} t + e_i + \sum_{l=1}^{p} c_i^l(s_i^l) + q_i^k + m_i^{j,k} & \text{if } u_i = j, v_i = k, j \neq k \\ t + e_i + \sum_{l=1}^{p} c_i^l(s_i^l) + q_i^j & \text{if } u_i = j, v_i = k, j = k \end{cases} \quad (3.4)$$

where t is the current time, e_i is the execution time of T_i, $\sum_{l-1}^{p} c_i^l(s_i^l)$ is the security overhead caused by p security services, q_i^k is the queuing time in the local task queue of the kth site in the system, and $m_i^{j,k}$ is the migration time if the task is migrated from the jth site to the kth site. Note that u_i denotes the site to which the task is submitted, and v_i represents the site where the task will be dispatched. e_i can be estimated by code profiling and statistical prediction [16]. q_i^k can be derived by task arrival rate and the execution time of all previously submitted tasks in the local task queue of the kth site [65]. $m_i^{j,k}$ is decided by two factors: (1) the size of data associated with task T_i that needs to be transferred from the jth site to the kth site; (2) the bandwidth between the two sites. The former is either the size of task T_i itself or the size of T_i plus the size of its input data. This information is provided by the user who submitted T_i, and thus, it can be known in advance. The latter can be obtained by exploiting certain network bandwidth prediction services like Network Weather Service [92].

If task T_i has a feasible schedule on a Grid represented as a set of sites, e.g., $N = \{N_1, N_2, \ldots, N_{j,\ldots}\}$, the following timing constraint must be satisfied, where d_i is the deadline of the task:

$$\exists N_j \in N, v_i = j : f_i \le d_i . \tag{3.5}$$

The total execution time of a workload with *n* tasks submitted to the system can be derived from Equations 3.6 and 3.7, meaning that the total execution time is the sum of the total overall execution times of admitted tasks running on each site in the Grid.

$$t_{exe} = \sum_{N_j \in N} t_{exe}^{j}, \tag{3.6}$$

$$t_{exe}^{j} = \sum_{i=1}^{n} a_i \left(e_i + \sum_{l=1}^{p} c_i^l (s_i^l) + q_i^j + b_i m_i^{u_i, v_i} \right), \tag{3.7}$$

where t_{exe}^{j} is the total execution time of tasks running on the *j*th site, and

$$a_i = \begin{cases} 0 \text{ if } v_i \neq j \text{ or } f_i > d_i \\ 1 \quad \text{otherwise} \end{cases}, \qquad b_i = \begin{cases} 0 \quad \text{if } u_i = j \\ 1 \quad \text{otherwise} \end{cases}$$

3.2 Summary

In this chapter, we presented a novel security middleware (SMW) model from which a security-sensitive real-time application can exploit a variety of security services to enhance the safety of its execution on clusters and Grids.

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

- Extend our SMW model to multi-dimensional computing resources. For now, we simply consider CPU time, which is only one of the computing resources consumed by the security services. Memory, network bandwidth and storage capacities should be considered in the future.
- Accommodate more security services into our SMW model. Besides the three security services discussed, we plan to take authorization and auditing services into consideration.

- Consider parallel applications where tasks have precedence constraints (see Chapter 6). In this chapter, we assume that all applications are independent. To make our scheme more practical, we need to extend it to handle general parallel applications.

Chapter 4

4 Security Overhead Model

In the previous chapter, we proposed a security-aware middleware model (SMW) from which security-sensitive real-time applications are allowed to exploit an array of security services to enhance trustworthy executions of the applications. It is essential for a security-aware scheduler, a critical component of the SMW model, to be aware of the amount of overhead caused by the security services. Otherwise, the security-aware scheduler is unable to make a correct schedule for real-time applications with security requirements. To reasonably measure security overhead and to make security-aware schedulers feasible, we present a security overhead model in this chapter.

This chapter is organized as follows. Section 4.1 gives an overview of the security overhead model. Sections 4.2 – Section 4.4 outline ways to quantify security overhead for confidentiality service, integrity service, and authentication service, respectively. Finally, Section 4.5 summarizes the security overhead model.

4.1 Quantitatively Measure Security Overhead

It is critical and fundamental to quantitatively measure overheads incurred by a wide range of security services, because security is achieved at the expense of performance. However, attention paid to models used to measure security overheads has been insufficient. Recently Irvine and Levin proposed a security overhead framework, which can be used for a variety of purposes [43]. Nevertheless, security overhead models for security services in the context of real-time computing remains an open issue. To enforce security in real-time applications while making security-aware scheduling algorithms predictable and practical, we propose in this section an effective model that is capable of approximately, yet reasonably, measuring security overheads experienced by tasks with security requirements. In light of the security overhead model, schedulers can incorporate security overheads into the process of scheduling tasks. Particularly, the model can be employed to compute the earliest start times and the minimal security overhead.

Without loss of generality, in this security overhead model we consider three security services widely deployed in clusters, namely, confidentiality, integrity, and authentication. We assume that the clusters are available, i.e., they respond tasks submitted by users. Please note that security mechanisms are not independent of one another. Rather, it is common that multiple security mechanisms are needed to form an integrated security solution, which can meet complex security demands. For example, authentication must be used in concert with message integrity. An array of primitive security services can be provided as building blocks for users to form integrated security solutions for applications. To examine the performance impact of each security service on our scheduling policies, we individually tested the three security services. This

experimental strategy by no means implies that in reality security services should be separated. The security overhead model (described in section 4.5) consists of the following three items (section 4.2-4.4).

4.2 Confidentiality Overhead

Encryption mechanisms support confidentiality by enciphering real-time applications (executable files) and data such that information and resources are not made available or disclosed to unauthorized persons or processes. Suppose there are eight encryption algorithms (see Table 4.1) deployed in a cluster. In accordance with the cryptographic algorithms' performance, each algorithm is assigned a security level in the range from 0 to 1. For example, we assign security level 1 to the strongest yet slowest encryption algorithm IDEA (see Table 4.1). Security levels for the rest algorithms can be computed by Equation 4.1, where μ_i^c is the performance of the ith ($1 \le i \le 8$) encryption algorithm.

$$sl_i^c = 13.5/\mu_i^c, 1 \le i \le 8. \tag{4.1}$$

Table 4.1 Cryptographic Algorithms for Confidentiality Service

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

Security levels of the algorithms are proportional to the algorithms' performance. Since computation overhead caused by encryption mainly depends on the cryptographic algorithms used and the size of data to be protected, Figure 4.1a shows encryption time in seconds as a function of encryption algorithms and size of secured data as measured on a 175 MHz Dec Alpha600 machine [56].

Let s_i^e be the confidentiality security level of task T_i, and the computation overhead of a selected confidentiality service can be calculated using Equation 4.2, where l_i is the amount of data whose confidentiality must be guaranteed, and $\sigma^c(s_i^c)$ is a function used to map a security level to its corresponding encryption method's performance.

$$c_i^c(s_i^c) = l_i / \sigma^c(s_i^c), 1 \le i \le 8. \tag{4.2}$$

4.3 Integrity Overhead

Integrity services ensure that no one can modify or tamper with data and applications while they are executing on clusters without being detected. Integrity can be accomplished by using a variety of hash functions [15]. Seven commonly used hash functions and their performance (evaluated on a 90 MHz Pentium machine) are shown in Table 4.2. Based on the hash functions' performance, each function is assigned a security level in the range from 0.18 to 1.0. We assign security level 1 to the strongest yet slowest hash function Tiger (see Table 2), and security levels for the other hash functions can be calculated by Equation 4.3, where μ_i^g is the performance of the ith ($1 \le i \le 7$) hash function.

$$sl_i^g = 4.36 / \mu_i^g, 1 \le i \le 7. \tag{4.3}$$

Let s_i^g be the integrity security level of task T_i, and the overhead of the integrity service can be calculated using Equation 4.4, where l_i is the amount of data whose integrity must be achieved, and $\sigma^g(s_i^g)$ is a function used to map a security level to its corresponding hash function's performance. The security overhead model for integrity is depicted in Figure 4.1b.

$$c_i^g(s_i^g) = l_i / \sigma^g(s_i^g), 1 \le i \le 7. \tag{4.4}$$

Table 4.2 Cryptographic Algorithms for Integrity Service

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

4.4 Authentication Overhead

It is of necessity that tasks are submitted from authenticated users and, therefore, authentication services are deployed to authenticate users who intend to access clusters [23] [28][37].

Table 4.3 illustrates three authentication techniques: weak authentication using HMAC-MD5; acceptable authentication using HMAC-SHA-1, fair authentication using CBC-MAC-AES. Each authentication technique is assigned a security level s_i^a in accordance with the performance. We assign security level 1 to the CBC-MAC-AES

method. Security levels for the other two methods can be obtained using Equation 4.5, where μ_i^a is the performance of the *i*th ($1 \le i \le 3$) authentication method.

$$sl_i^a = \mu_i^a / 163, 1 \le i \le 3. \quad (4.5)$$

Table 4.3 Cryptographic Algorithms for Authentication Service

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

Authentication overhead $c_i^a(s_i^a)$ of task T_i is a function of T_i's security level s_i^a. The security overhead model for authentication is shown in Figure 4.1c.

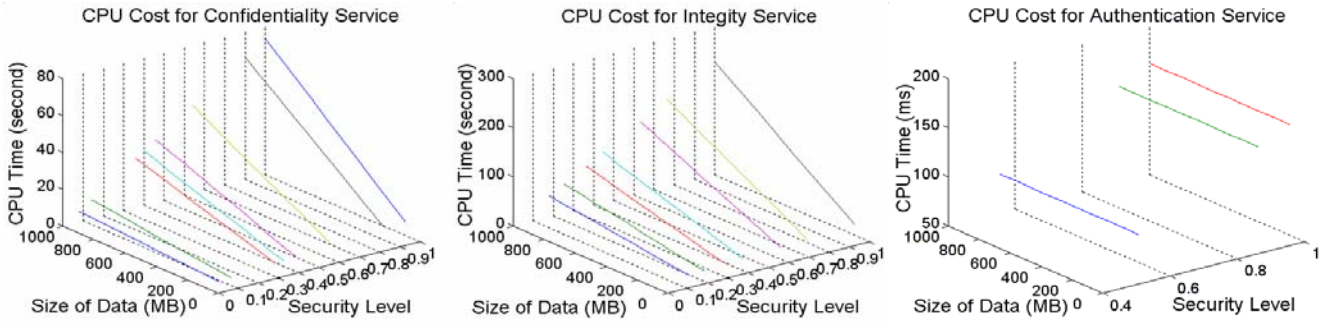


Figure 4.1 Security overhead model

4.5 Summary

We can derive security overhead, which is the sum of the overheads imposed by all involved security services. Suppose task T_i requires q security services provided in sequential order. Let s_i^j and $c_i^j(s_i^j)$ be the security level and overhead of the *j*th security service, the security overhead c_i experienced by T_i, can be computed using Equation 4.6.

In particular, the security overhead of T_i with security requirements for the three services above is measured by Equation 4.7.

$$c_i = \sum_{j=1}^{q} c_i^j(s_i^j), \text{ where } s_i^j \in S_i^j. \tag{4.6}$$

$$c_i = \sum_{j\in\{a,c,g\}} c_i^j(s_i^j), \text{where } s_i^j \in S_i^j. \tag{4.7}$$

Noted that $c_i^c(s_i^c)$, $c_i^g(s_i^g)$, and $c_i^a(s_i^a)$ in Equation 4.7 are derived from Equation 4.2, Equation 4.4 and Table 4.3. Equation 4.7 will be applied to calculated the earliest start times and minimal security overhead.

How to quantitatively measure security strength is a hard and open issue to be solved by computer security community. In this dissertation research we assign a security level to a security mechanism based on its performance. This security level calculation method is based on an assumption that people only accept a slower security mechanism if and only if it can provide a higher level security compared with its faster peers. This assumption is theoretically correct but in reality some slow yet not very secure encryption algorithms like DES are still being used due to all kinds of reasons. The security overhead model discussed in this chapter only for illustration purpose as quantitatively measuring security strength of various security mechanisms is out of scope of this dissertation research.

Chapter 5

5 Security-Aware Scheduling for Real-Time Applications on Homogeneous Clusters

In the previous chapter, we proposed a security overhead model, which can be used to calculate security overhead caused by security requirements of real-time applications. With this model in place, in this chapter we propose a security-aware real-time heuristic strategy (SAREC), which integrates security requirements into the scheduling for real-time independent tasks on homogeneous clusters.

This chapter is organized as follows. Section 5.1 presents the motivation of this study. Security and real-time requirements of real-time tasks are modeled in Section 5.2. In Section 5.3, the SAEDF algorithm, which is a combination of SAREC and EDF (Earliest Deadline First), is described. Empirical results based on a number of simulated homogeneous clusters are discussed in Section 5.4. Section 5.5 provides a summary of this chapter.

5.1 Motivation

Security-critical real-time applications such as military aircraft flight control systems have mandatory security requirements in addition to stringent timing constraints. Conventional real-time scheduling algorithms, however, either disregard applications' security needs and thus expose the applications to security threats, or run applications at inferior security levels without optimizing security performance.

In the last decade, clusters have become increasingly popular as powerful and cost-effective platforms for executing parallel applications [61][62]. Much of this trend can be attributed to rapid advances in processing power, network bandwidth, and storage capacity. In recognition that many applications running on clusters demand both real-time performance and security, we investigate the problem of scheduling a set of independent real-time tasks with various security requirements. In the previous chapter, we have presented a novel security overhead model that can be used to reasonably measure security overheads incurred by the security-critical tasks. Now we are positioned to propose a security-aware real-time heuristic strategy for clusters (SAREC). To evaluate the performance of SAREC, we incorporate the earliest deadline first (EDF) scheduling policy into SAREC to implement a new security-aware real-time scheduling algorithm (SAEDF).

5.2 Security and Real-Time Requirements

5.2.1 Security-Aware Scheduling Architecture

We focus in this study on an *m*-node cluster in which *m* identical nodes are connected via a high-speed network, e.g., Myrinet and Fast Ethernet, to process soft real-time tasks

submitted by r users. Let $N = \{N_1, N_2, \ldots, N_m\}$ denote a set of identical computational nodes. The architecture of security-aware real-time scheduling shown in Figure 5.1 encompasses the SAREC strategy and a real-time scheduler. The SAREC strategy is implemented in form of a security level controller and an admission controller. In this study we build the real-time scheduler using the EDF policy, which can be substituted by other real-time scheduling policies. The admission controller determines if an arriving task in a *schedule queue* can be accepted or not, whereas the security level controller aims at maximizing the security levels of admitted tasks.

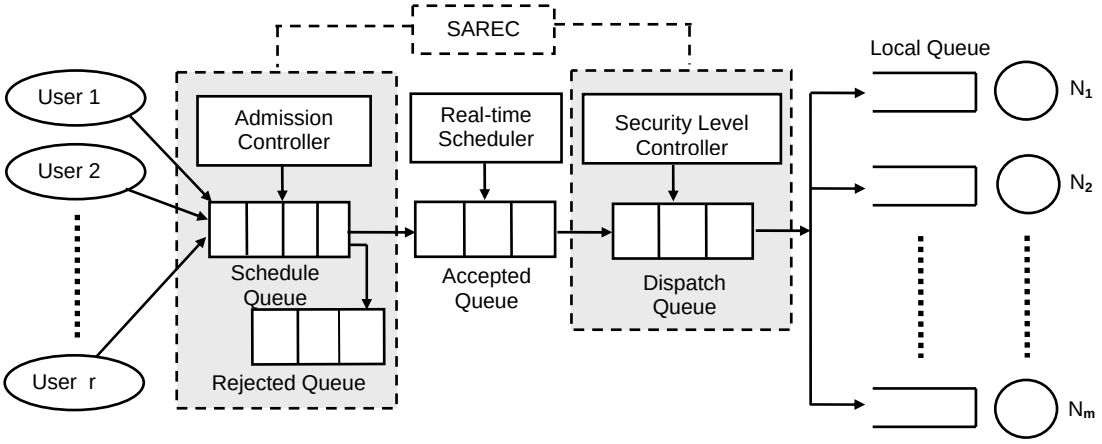


Figure 5.1 Security-aware real-time scheduling architecture

The schedule queue maintained by the admission controller is deployed to accommodate incoming real-time tasks. If the deadline and minimal security requirements of an incoming task can be guaranteed, the admission controller will place the task in an *accepted queue* for further processing. Otherwise, the task will be dropped into a *rejected queue*. The real-time scheduler processes all the accepted tasks by its scheduling policy before the tasks are transmitted into a *dispatch queue*, where the security level controller escalates the security level of the first task under two conditions: (1) the security level promotion will not make the first task miss its deadline; and (2)

increasing the security level will not make any previously accepted task miss its deadline. After being handled by the security level controller, the task is dispatched to one of the designated node $N_i \in N$ referred to as a *processing node* for execution. Each processing node maintains a *local queue*.

5.2.2 Real-Time Tasks with Security Requirements

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

A task T_i submitted by a user is modeled as a set of rational parameters, e.g., $T_i = (a_i, e_i, f_i, d_i, l_i, S_i)$, where a_i, e_i, and f_i are the arrival, execution, and finish times, d_i is the deadline, and l_i denotes the amount of data (measured in KB) to be protected. e_i can be estimated by code profiling and statistical prediction [16]. Suppose T_i requires q security services represented by a vector of security level ranges, e.g., $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$. The vector characterizes the security requirements of the task. S_i^j is the security level range of the jth security service required by T_i. The security level controller determines the most appropriate point s_i in space S_i, e.g., $s_i = (s_i^1, s_i^2, \ldots, s_i^q)$, where $s_i^j \in S_i^j$, $1 \le j \le q$.

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

$$SL(s_i) = \sum_{j=1}^{q} w_i^j s_i^j \ , \ 0 \le w_i^j \le 1, \ \sum_{j=1}^{q} w_i^j = 1 \ . \tag{5.1}$$

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

X_i denotes all possible schedules for task T_i, and $x_i \in X_i$ be a scheduling decision of T_i. x_i is a feasible schedule if (1) deadline d_i can be guaranteed, i.e., $f_i \le d_i$, and (2) the security requirements are met, i.e., $\min(S_i^j) \le s_i^j \le \max(S_i^j)$. Given a real-time task T_i, the security benefit of T_i, is expected to be maximized by the security level controller (See Figure 5.1) under the timing constraint:

$$\begin{aligned} SB(X_i) &= \max_{x_i \in X_i} \{SL(s_i(x_i))\} \\ &= \max_{x_i \in X_i} \left\{ \sum_{j=1}^{q} w_i^j s_i^j(x_i) \right\}, \end{aligned} \tag{5.2}$$

where the security level of the jth service $s_i^j(x_i)$ is obtained under schedule x_i, and $\min(S_i^j) \le s_i^j(x_i) \le \max(S_i^j)$. $\min(S_i^j)$ and $\max(S_i^j)$ are the minimum and maximum security requirements of task T_i.

A security-aware scheduler aims at maximizing the system's quality of security, or security value, defined by the sum of the security levels of admitted tasks (See Equation 5.1). Thus, the following security value function needs to be maximized, subject to certain timing and security constraints:

$$SV(X) = \max_{x \in X} \left\{ \sum_{i=1}^{p} y_i SB(x_i) \right\}, \tag{5.3}$$

where p is the number of submitted tasks, y_i is set to 1 if task T_i is accepted, and is set to 0 otherwise. Substituting Equation 5.2 into 5.3 yields the following security value objective function. Our proposed security-aware scheduling algorithm strives to schedule tasks in a way to maximize Equation 5.4:

$$SV(X) = \max_{x \in X} \left\{ \sum_{i=1}^{p} \left(y_i \max_{x_i \in X_i} \left\{ \sum_{j=1}^{q} w_i^j s_i^j (x_i) \right\} \right) \right\}. \tag{5.4}$$

5.3 The SAEDF Algorithm

In Section 5.2 we proposed the SAREC strategy. Now we evaluate the effectiveness of SAREC by proposing a novel security-aware real-time scheduling algorithm, SAEDF (Security-Aware EDF), which incorporates the earliest deadline first (EDF) scheduling algorithm into the SAREC strategy.

To support the presentation of the proposed algorithm, it is necessary to introduce three properties. The schedule of a task is feasible if the task is completed before its deadline. Hence, a task has a feasible schedule on a cluster if there exists at least one node, where a valid schedule is available for the task. More formally, this fact can be expressed by the following property.

Property 5.1. If task Ti has a feasible schedule on a cluster with m nodes denoted by a set N = {N1, N2, …, Nm}, the following inequality must be satisfied:

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

$\forall T_k \in N_j, d_k > d_i : \mathrm{es}_j(T_k) + e_i + c_i^{min} \le d_k$, where esj(Ti) is the earliest start time of task Ti on node N2, ei and diare the execution time and deadline of Ti, and c_i^{min} is the security

overhead experienced by Ti when its minimal security requirements are met. The condition enforced in Property 5.1 indicates that the execution of Ti on Nj results in no violation of any deadlines of tasks that have been admitted to the cluster.

The earliest start time $\mathrm{es}_j(T_i)$ can be computed by Equation 5.5.

$$\mathrm{es}_j(T_i) = r_j + \sum_{T_k \in N_j, d_k \le d_i} \left(e_k + \sum_{l \in \{a,c,g\}} c_k^l(s_k^l) \right), \tag{5.5}$$

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

The minimal security overhead c_i^{min} of T_i can be calculated by the following equation.

$$c_i^{min} = \sum_{j \in \{a,c,g\}} c_i^j \left(\min \left\{ S_i^j \right\} \right), \tag{5.6}$$

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

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

Property 5.2. Task T_i meets its deadline. Thus, $\mathrm{es}_j(T_i) + e_i + \sum_{j \in \{a,c,g\}} c_i^j(s_i^j) \le d_i$, where $s_i^j \in S_i^j$ is the security level of the jth security service.

Property 5.3. The security level of an accepted task T_i on node N_j is maximized at the task's arrival time under the assumption that no more tasks arrive on N_j after this arrival time.

1.for each task T_i submitted to the schedule queue do
2. for each node N_j in the cluster do
3. Use Equation 12 to compute $es_j(T_i)$, the earliest start time of task T_i on node N_j;
4. Use Equation 13 to obtain the minimal security overhead c_i^{min} of task T_i;
5. if $\text{es}_j(T_i) + e_i + c_i^{min} \le d_i$ and $\exists T_k \in N_j, d_k > d_i$: $\text{es}_j(T_k) + e_i + c_i(N_j) \le d_k$
6. Sort the security service weights in a decreasing order of their values, e.g., $w_i^{v_1} < w_i^{v_2} < w_i^{v_3}$, where $v_l \in \{a, c, g\}, 1 \le l \le 3$;
7. for each security service $v_l \in \{a, c, g\}, 1 \le l \le 3,$ do
8. $s_i^{v_l} = \min\{S_i^{v_l}\}$; /* Initialize the security value of security service v_l */
9. end for
10. for each security service $v_l \in \{a, c, g\}, 1 \le l \le 3,$ do
11. while $s_i^{v_l} < \max\{S_i^{v_l}\}$ do
12. Increase security level $s_i^{v_l}$;
13. Use Equation 11 to calculate security overhead $c_i(N_j)$ of T_i on N_j;
14a. if $es_j(T_i)+e_i+c_i > d_i$ (Property 2)
14b. $\exists T_k \in N_j, d_k > d_i$: $es_j(T_k)+e_i+c_i(N_j) > d_k$ (Property 1) then
15. decrease security level $s_i^{v_l}$; break;
16. end while
17. end for
18. $SL_i^j \leftarrow SL(s_i)$; /* Obtain the security level of T_i on N_j using Equation 1 */
19. else $SL_i^j \leftarrow 0$; /* Set the security level to 0 because T_i has no feasible schedule on N_j */
20. end for
21. if $\exists N_j \in N : SL_i^j > 0$ then
22. $y_i \leftarrow 1$; /* Accept task T_i */
23. /* Optimize quality of security, see Equation 2 */
Find node N_k for T_i, subject to: $SL_i^k = \max_{1 \le j \le n}\{SL_i^j\}$;
24. dispatch task T_i to N_k according to the schedule generated above;
25. else $y_i \leftarrow 0$/* Reject T_i, since no feasible schedule is available */
26.end for

Figure 5.2 The SAEDF algorithm

The SAEDF algorithm is outlined in Figure 5.2. The goal of the algorithm is to deliver high quality of security while guaranteeing real-time requirements for tasks running on clusters. To achieve the goal, SAEDF strives to maximize security level (see

Equation 5.1) of each accepted task (see Step 23) while maintaining reasonably high guarantee ratios (see Step 5).

Before optimizing the security level of task T_i on N_j, SAEDF attempts to meet the real-time requirement of T_i. This can be accomplished by calculating the earliest start time (see Equation 5.5) and the minimal security overhead of T_i (see Equation 5.6) in Steps 3 and 4. Next, Step 5 checks if the cluster can meet the timing constraints of T_i and all tasks whose deadlines are later than that of T_i. If the timing constraints can not be satisfied, Step 19 sets T_i' security level on N_j to 0, indicating that T_i can not be allocated to node N_j. In case no node the cluster can produce a feasible schedule for T_i, it is rejected by Step 25.

The security level of T_i on N_j is optimized in the following way. The security service weights used in Equations 5.1 and 5.2 reflect the importance of the three security services, indicating that it is desirable to give higher priorities to security services with higher weights (see Step 6). In other words, enhancing security levels of more important services tends to yield a maximized security level of T_i on N_j.

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

Now we evaluate the time complexity of SAEDF as follows.

Theorem 5.1. *The time complexity of SAEDF is O(knm), where m is the number of nodes in the cluster, n is the number of tasks in the local queue of a node, and k is the number of possible security level ranks for a particular security service* v_l $(v_l \in \{a, c, g\}, 1 \le l \le 3)$.

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

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

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

Theorem 5.2. *The SAEDF algorithm satisfies Properties 2 and 3.*

Proof. (1) First, we prove that SAEDF satisfies Property 5.2. A task T_i is accepted by a cluster with *m* nodes denoted by $N = \{N_1, N_2, \ldots, N_m\} \Rightarrow$ There is at least one node N_j ($N_j \in N$) on which T_i has a feasible schedule $\overset{\text{Property 1}}{\Rightarrow}$ The two inequalities in Property 5.1 must

hold $\overset{inequality\,1}{\Rightarrow}$ task T_i can be finished before its deadline $d_i \Rightarrow$ The deadline of task T_i must be met. Thus, each accepted task meets its deadline.

(2) Second, we prove that SAEDF satisfies Property 5.3. We provide a proof by contradiction. There are two cases after task T_i is accepted:

(a) Task T_i is the last element in the local queue of node N_j based on the EDF order. In this case, there is no other task in the local queue of node N_j which is behind T_i. The only constraint for increasing the security level of T_i is its deadline d_i, which is enforced by Step 14a in Figure 5.2. The security level of task T_i will eventually reach a critical value SL_i^{jc1} (Steps 10 ~ 18 in Figure 5.2), meaning that any further increase in security level of T_i will violate its deadline d_i. Now suppose that there is a higher security level SL_i^{jb1} ($SL_i^{jb1} > SL_i^{jc1}$) for task T_i which is an accepted task on node N_j. However, this SL_i^{jb1} definitely makes T_i violate its deadline d_i based on the conclusion drawn above because of the equality $SL_i^{jb1} > SL_i^{jc1}$. SL_i^{jb1} makes T_i miss its deadline $d_i \Rightarrow \text{es}_j(T_i) + e_i + c_i > d_i \Rightarrow T_i$ cannot be accepted by node $N_j \Rightarrow$ This statement contradicts our assumption that task T_i is an accepted task on N_j. Thus, SL_i^{jc1} must be the maximal security level of T_i under this situation.

(b) Task T_i is not the last element in the local queue of node N_j based on the EDF order. Thus, there exists at least one previously accepted task to be executed after T_i is finished. The timing constraint is enforced by Step 14a. The security level of task T_i will also eventually reach a critical value SL_i^{jc2} (Steps 10 ~ 18 in Figure 5.2), which means that further increase in the security level of T_i will either violate T_i's deadline or the deadlines of earlier accepted tasks. Now suppose SL_i^{jc2} is not T_i's maximal security level under this

circumstance and, thus, there is a larger security level SL_i^{jb2} ($SL_i^{jb2} > SL_i^{jc2}$) for task T_i, an accepted task on node N_j under this situation. However, SL_i^{jb2} will violate either deadline d_i or the deadlines of earlier accepted tasks because of the inequality $SL_i^{jb2} > SL_i^{jc2}$.

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

Case two: SL_i^{jb2} violates the deadlines of earlier accepted tasks. Thus, $\exists T_k \in N_j$ $d_k > d_i$: $es_j(T_k)+e_k+c_k^{min}(N_j) > d_k$. The implication is that the second inequality in Property 5.1 does not hold. Therefore, task T_i has no feasible schedule on node N_j, meaning that T_i is not an accepted task on node N_j. This statement contradicts our assumption that T_i is an accepted task on node N_j. Consequently, SL_i^{jc2} must be the maximal security level of T_i under this situation. □

5.4 Experimental Results

We evaluate in this section the performance of the SAEDF algorithm using extensive simulation experiments based on real world traces consisting of 29695 tasks. A competitive advantage of conducting simulation experiments is that performance evaluation on a large-scale cluster can be accomplished without additional hardware cost. To reveal performance improvements gained by our proposed algorithm, we compare SAEDF with three well-known scheduling algorithms, namely, EDF (Earliest Deadline First) [77], LLF (Least Laxity First)[51], and FCFS (First Come First Serve). To make the comparisons fair, we slightly modify the three algorithms in a way that they arbitrarily pick a security level within the security level range of each service required by

a task. Although these algorithms are intended to schedule real-time tasks with security requirements, they make no effort to optimize quality of security. The baseline algorithms are briefly described below.

1. EDF: The task with the earliest deadline is always executed first.

2. LLF: The task with the minimal laxity (slack time) is always executed first.

3. FCFS: Tasks will be executed in the non-decreasing order of their arrival times.

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

5.4.1 Simulator and Simulation Parameters

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

We modified the traces used in [38][105] by adding randomly generated deadlines for all tasks in the traces, which were collected from one workstation on six different time intervals. The assignment of deadlines is controlled by the deadline base (Tbase) denoted as β, which sets an upper bound on tasks' slack times. We use Equation 5.7 to generate T_i's deadline d_i.

$$d_i = a_i + e_i + c_i^{max} + \beta, \tag{5.7}$$

where a_i and e_i are the arrival and execution times obtained from the real-world traces. c_i^{max} is the maximal security overhead (measured in ms), which is computed by Equation 5.8.

$$c_i^{\max} = \sum_{j \in \{a,c,g\}} c_i^j \left(\max\left\{S_i^j\right\}\right), \tag{5.8}$$

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

Table 5.1 Characteristics of system parameters

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

Although CPU demands of tasks submitted to the clusters are taken directly from the existing traces, deadlines are synthetically generated in accordance with the above model. The simplification weakens correlations between real-time requirements and other workload characteristics. However, in the experiments we can examine impacts of deadlines on system performance by controlling the deadlines as fundamental simulation parameters (see Section 5.4.2). Similarly, each task was synthetically assigned a block of data that needs to be protected from being disclosed or tampered with. The impact of security-required data size is examined in Section 5.4.7. The performance metrics by which we evaluate system performance include: *security value* (*SV*, see Equation 5.4), *guarantee ratio* (*GR*, measured as a fraction of total submitted tasks that are found to be schedulable), and *overall system performance* (*OSP*, defined as a product of normalized security value and guarantee ratio, see Equation 5.9), where

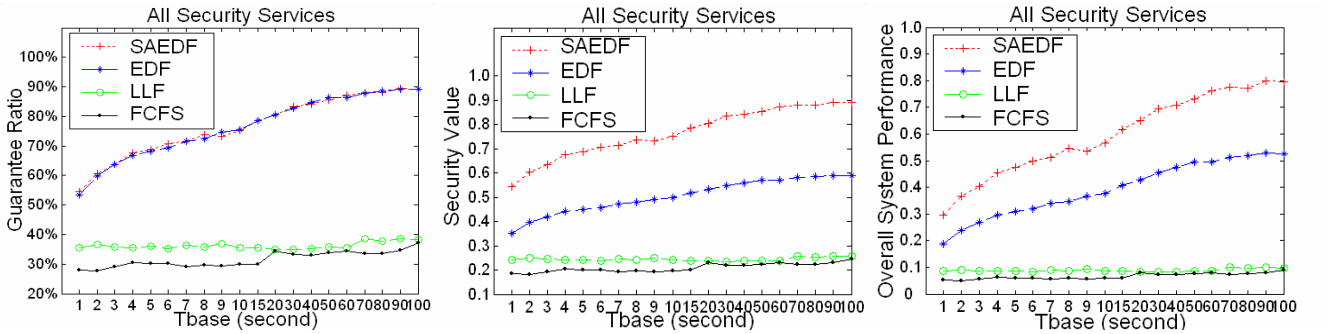
$$OSP = GR * SV \tag{5.9}$$

5.4.2 Overall Performance Comparisons

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

Figure 5.3 shows the simulation results for these four algorithms on a cluster with 64 nodes. We observe from Figure 5.3a that SAEDF and EDF exhibit similar performance in

terms of guarantee ratio, whereas SAEDF noticeably outperforms LLF and FCFS algorithms. Although LLF is a real-time scheduling algorithm, it does not favour short tasks as EDF does. Therefore, many subsequent short tasks are likely to miss their deadlines due to the acceptance of long tasks. FCFS has the lowest guarantee ratios, because FCFS is a non-real-time scheduling policy. It is observed that SAEDF and EDF maintain high guarantee ratios. We attribute the guarantee ratio improvement of SAEDF over LLF and FCFS to the fact that SAEDF judiciously boosts the security levels of accepted tasks under the condition that timing constraints are met.



(a) **(b)** **(c)**

Figure 5.3 Performance of the four scheduling algorithms

Figure 5.3b plots security values of the four algorithms when the deadline base is increased from 1 to 100 seconds. Figure 5.3b reveals that SAEDF consistently performs better, with respect to quality of security, than all the rest approaches. Specifically, SAEDF outperforms EDF, LLF, and FCFS in security value by averages of 43.6%, 248.9%, and 266.7%, respectively. Interestingly, when the deadlines become loose, the performance improvements of SAEDF over the three competitors are more pronounced. This is because the SAEDF approach is capable of employing slack times to improve the quality of security of accepted tasks. Therefore, the more slack time available, the higher

security value can be achieved. The results clearly indicate that clusters can gain more performance benefits from the SAEDF algorithm under workload conditions where real-time tasks have loose deadlines.

Figure 5.3c plots the overall system performance improvements achieved by SAEDF. An observation made from Figure 5.3c is that SAEDF significantly outperforms all the other three alternatives. This can be explained by the fact that, although the guarantee ratios of SAEDF and EDF are similar, SAEDF considerably improves security values over the other algorithms, while achieving higher guarantee ratio than LLF and FCFS. The result suggests that if quality of security is the sole objective in scheduling, SAEDF is more suitable for clusters than the other algorithms. In contrast, if schedulability is the only performance objective, SAEDF can maintain the same guarantee ratios as those of EDF, which is inferior to SAEDF in terms of security.

5.4.3 Impact of the Security Overhead Model

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

Figure 5.4 – Figure 5.6 show the performance impacts of the authentication, confidentiality, and integrity services, respectively. We observe from the figures that SAEDF delivers better overall system performance than the other competitors under a

wide range of workload conditions. This result is consistent with that observed from the previous experiments (see Figure 5.3), where each task requires all three security services.

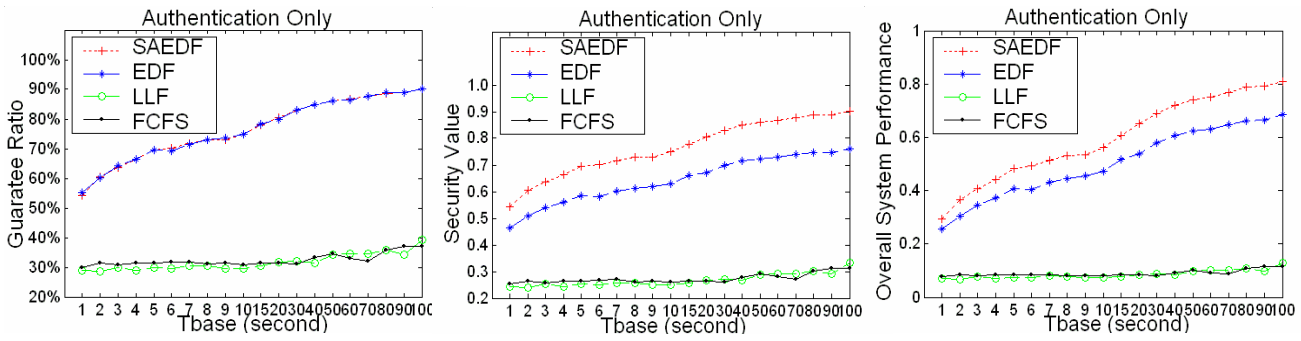


Figure 5.4 Performance impact of authentication security service

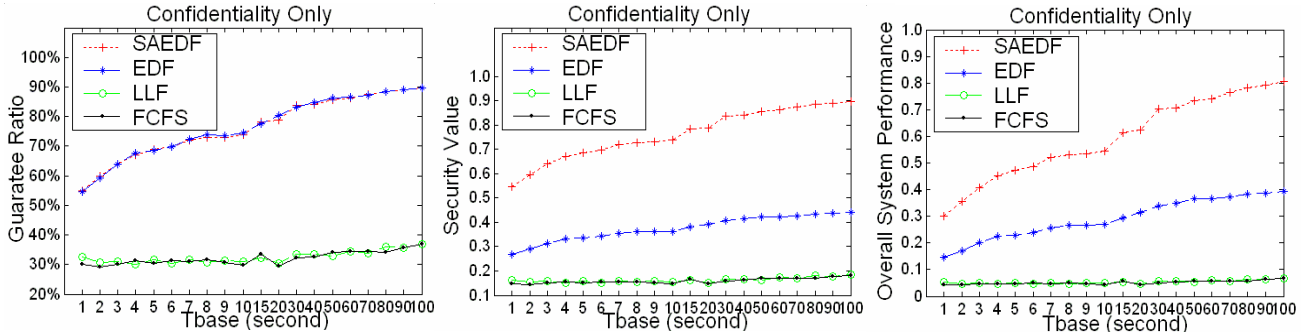


Figure 5.5 Performance impact of confidentiality security service

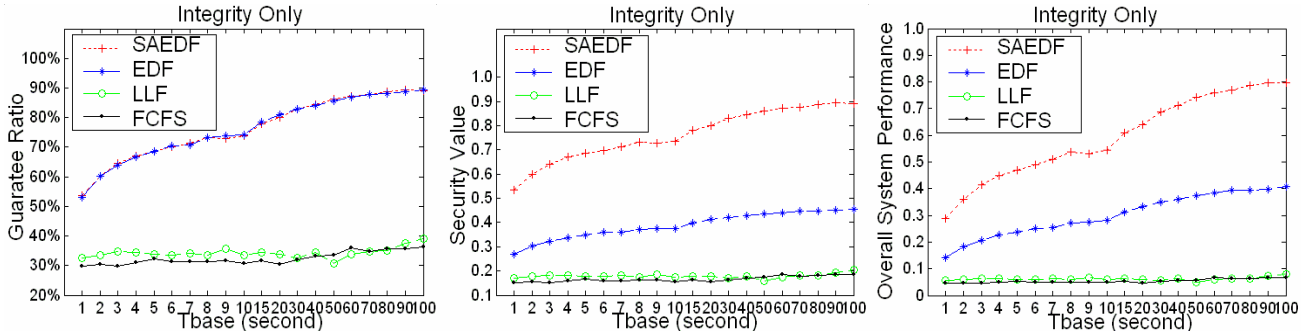


Figure 5.6 Performance impact of integrity security service

Interestingly, the security improvements are more pronounced when the confidentiality or integrity service is required than when the authentication service is

needed. The reason is three-fold. First, there are only three security levels for the authentication service in the security overhead model, and the granularity of security levels for authentication is coarser than those of the confidentiality and integrity services. Second, the authentication overhead is less than that of the confidentiality and integrity services in most cases. Thus, it is relatively easy to achieve a higher security level in the authentication service for an accepted task. Third, the confidentiality and integrity overheads rely on the amount of data to be protected, whereas the authentication overhead is independent of the security-required data size.

5.4.4 Impact of Security Service Weights

Recall that the security level model proposed in Section 5.2.2 is comprised of multiple security levels for a diversity of security services like confidentiality, integrity, and authentication. Each service required by a task is assigned a weight, which reflects the priority of the service. To study the impact of security service weights on performance of SAEDF, we set the authentication weight to a constant value, and varied the confidentiality and integrity weights. Specifically, Figure 5.7 plots the performances of the four algorithms when the confidentiality weight is increased from 0.1 to 0.8, whereas Figure 5.8 depicts the performances when the confidentiality weight varies from 0.1 to 0.6.

The first observation drawn from Figure 5.7 and Figure 5.8 is that for all the algorithms, the performance in guarantee ratio is independent of the security service weights. The implication of this result is that the security service weights are irrelevant to overall execution times of tasks. The second intriguing observation made from Figure 5.7 and Figure 5.8 is that the confidentiality and integrity weights slightly affect the security

performance of SAEDF, while making considerable impact on the other three algorithms in terms of security value. This is because at the same security level, confidentiality service overhead is relatively smaller than integrity service overhead. Consequently, the overall security values of accepted tasks tend to increase when the confidentiality weight goes up. These results indicate that SAEDF can marginally improve security performance for workloads where confidentiality service is more important than the other concerns.

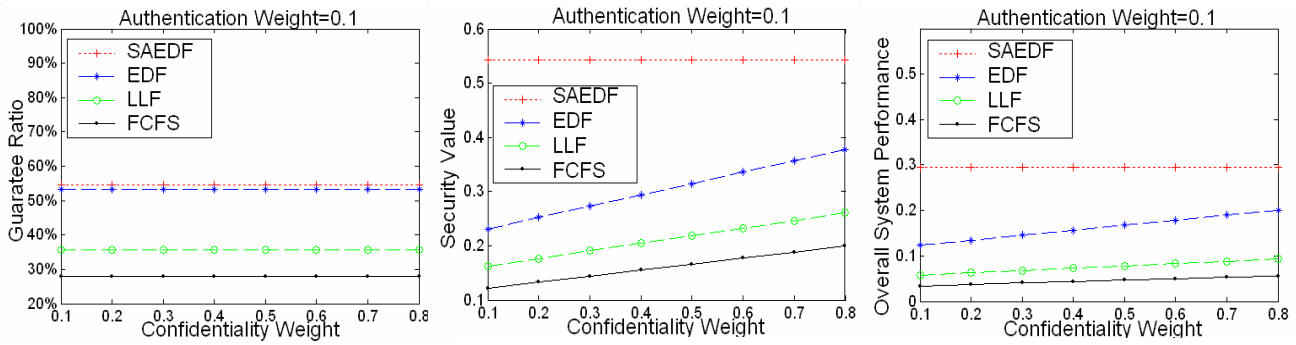


Figure 5.7 Performance impact of security service weights, authentication weight = 0.1

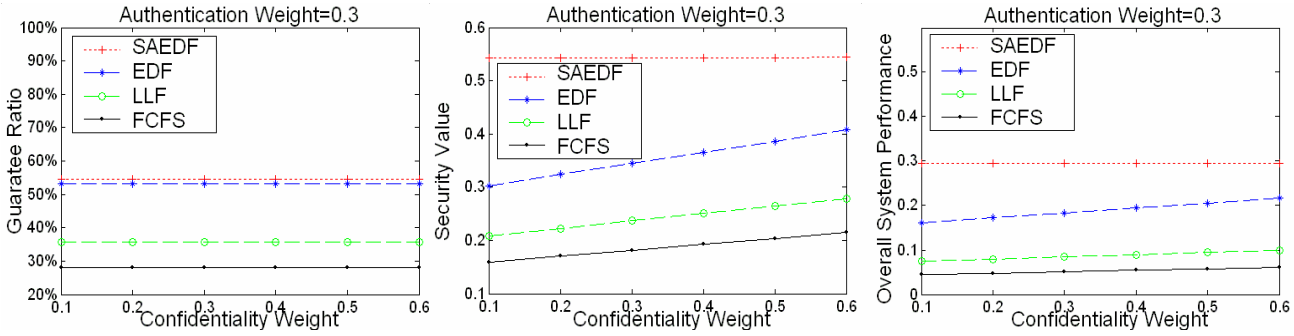


Figure 5.8 Performance impact of security service weights, authentication weight = 0.3

5.4.5 Sensitivities to CPU Capacity

To examine performance sensitivities of the four algorithms to CPU capacity, in this set of experiments we varied the CPU capacity from 100 to 800 MIPS with increments of

100 MIPS.

The results reported in Figure 5.9 reveal that SAEDF outperforms the other three alternatives in terms of security value and overall system performance. With respect to guarantee ratio, SAEDF exhibits a similar performance to EDF and LLF. The guarantee ratio of FCFS even decreases when CPU capacity enlarges. This is mainly because tasks with long execution times can be admitted when the CPU capacity is high and, therefore, there is a strong likelihood for more small tasks to miss their deadlines.

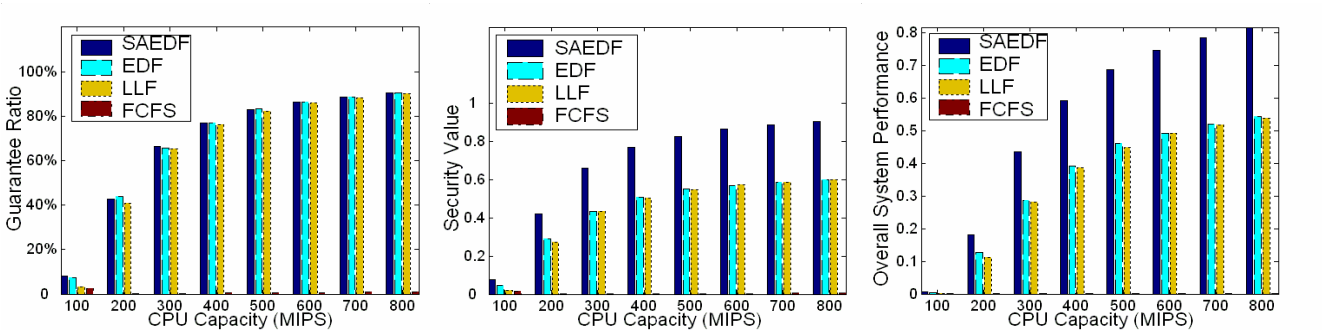


Figure 5.9 Performance sensitivities to CPU capacity

5.4.6 Scalability

This experiment is intended to investigate the scalability of the SAEDF algorithm. We scale the number of nodes in a cluster from 8 to 256. Figure 5.10 plots the performances as functions of the number of nodes in the cluster. It is observed from Figure 10 that the amount of improvement achieved by SAEDF becomes more prominent with the increasing value of the node number. This result shows that the SAEDF approach exhibits good scalability.

Figure 5.11 shows the improvements of SAEDF in overall system performance over the other three policies. SAEDF outperforms the three baseline algorithms in terms of overall system performance by averages of 70.4%, 201.2%, and 625.6%, respectively.

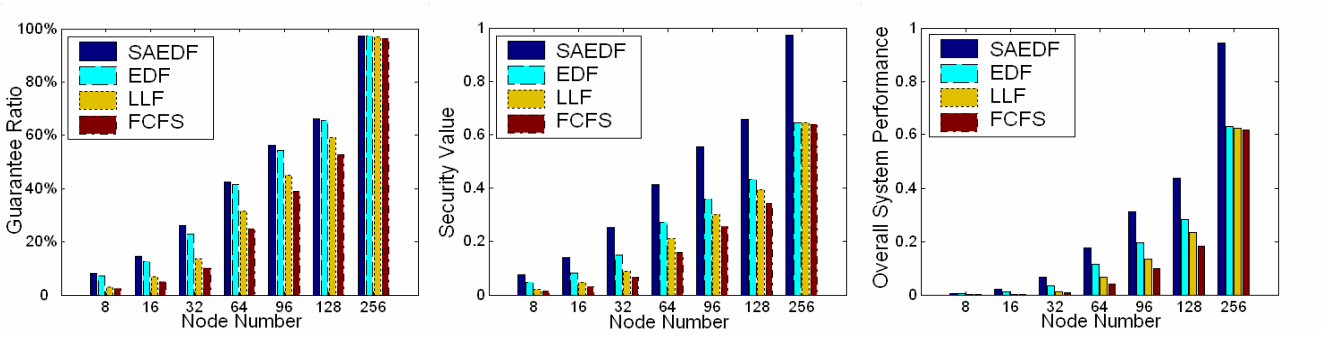


Figure 5.10 Scalabilities of the four scheduling algorithms

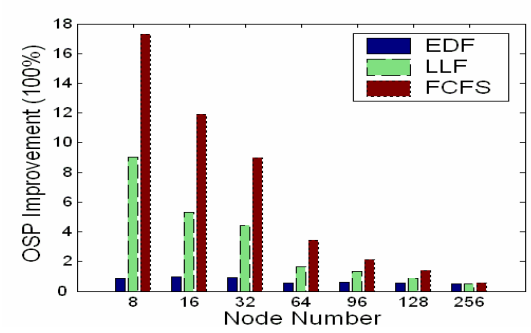


Figure 5.11 Overall system performance improvement

5.4.7 Security-Required Data Size

In this set of experiments we evaluated the performance impact of security-required data size. We tested three configurations of data size (see Table 5.1). The laxity is chosen to be 1000 millisecond. Without loss of generality, we assume that the distribution of the data size is a normal distribution. The mean size of the security-required data varies from 50 KB to 4 MB and the standard deviation changes from 40 to 20000. For example, in config1, the mean size is 50KB for short tasks, 500KB for middle tasks and 1MB for long tasks. The standard deviation is set to 40 for short tasks and set to 20000 for medium and long tasks.

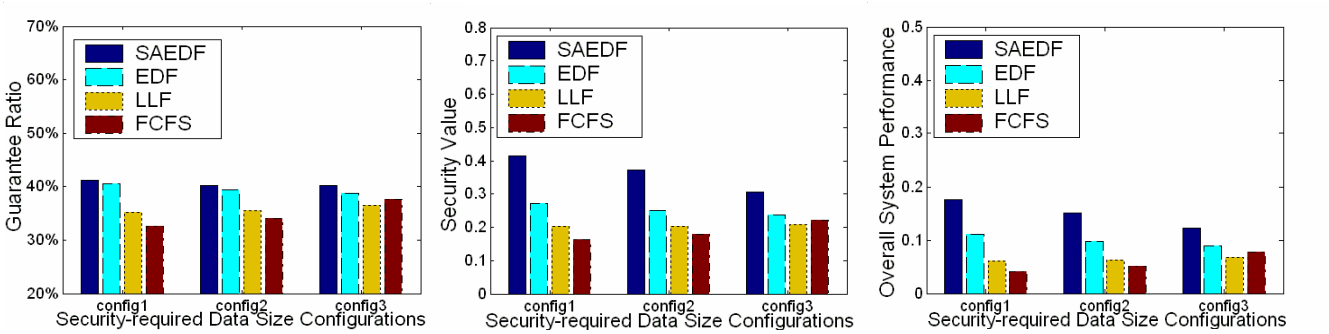


Figure 5.12 Performance impact of size of data to be secured

There are several important observations that can be drawn from Figure 5.12. First, when the security-required data size increases, the guarantee ratio of SAEDF almost remains unchanged, while SAEDF's security value drops. This phenomenon reveals that SAEDF is a security-aware algorithm, which judiciously lowers accepted tasks' security levels under heavily loaded conditions in order to accommodate more tasks. Unlike SAEDF, the guarantee ratio of EDF noticeably decreases with the increasing size of security-required data. Second, Figure 5.12 shows that the guarantee ratios of LLF and FCFS increase with the growing size of security-required data. This is because large tasks are more likely to be dropped due to their high security overhead caused by enlarged security-required data size. As a result, a vast majority of the small tasks submitted to the cluster can be finished before their deadlines.

5.4.8 Integrate SAREC into LLF

To demonstrate that SAREC is a general security-aware strategy that can be incorporated into other existing real-time scheduling algorithms, we integrate SAREC with the least-laxity-first algorithm (LLF) [51] to construct a new algorithm called SALLF (Security-Aware LLF). Now we evaluate the performance of SALLF in this subsection.

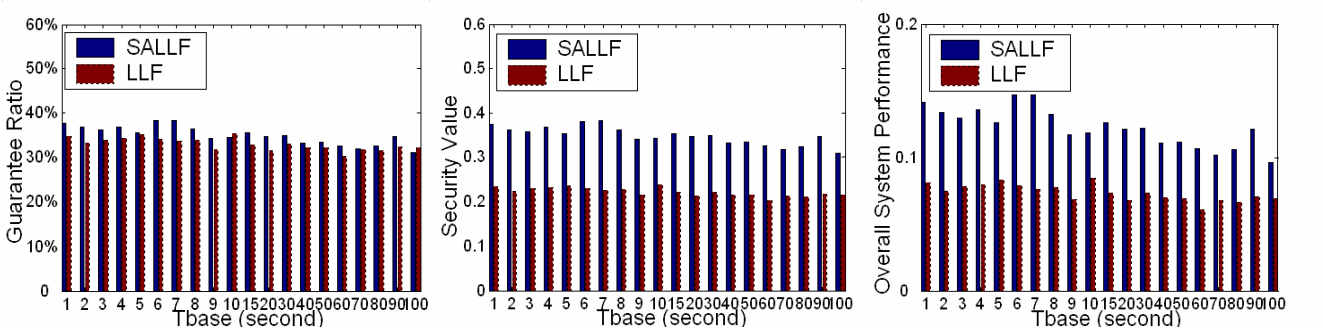


Figure 5.13 Performance improvement of SALLF over LLF

One important observation from Figure 5.13 is that SALLF outperforms LLF in nearly all cases. Specifically, SALLF improves guarantee ratio over LLF by an average of 6.1% and outperforms LLF in terms of security value by an average of 55.8%. The rationale behind these results is that SALLF can maximize guarantee ratios by adaptively adjusting tasks' security levels, while LLF has no capability of optimizing security levels.

5.4.9 A Real Application – Aircraft Flight Control

To validate the results from the trace-driven simulations, we applied our SAEDF algorithm to a real world system – an automated flight control system [2]. Table 5.2 shows the set of parameters present for all real-time tasks, including execution time, period, and three configurations of size of data to be secured.

The automated flight control system was utilized to fly a simulated model of an F-16 fighter aircraft. Details of the automated flight control system can be found in [50][54]. In this system all the flight control tasks - including Guidance, Slow Navigation, Fast Navigation, Controller and Missile Control - need to be executed in real-time to meet their deadlines. The functions of these five tasks are summarized as follows [2]: The

"Guidance" task sets the reference trajectory of the aircraft in terms of altitude and heading; the "Controller" is responsible for executing the closed-loop control functions that deal with actuator commands; the two "Navigation" tasks read sensor values distinguished by the required update frequency; and finally, the "Missile Control" task is responsible for reading radar and firing missiles. These separate tasks are mandatory to control the aircraft during flight, and they are all cyclic tasks with multiple versions.

Table 5.2 Parameters for Automated Flight Control System

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

It is assumed that (1) all tasks have known bounded execution times, (2) task arrivals are independent, (3) each task's deadline is equal to its period, and (4) tasks are non-preemptive in nature [2]. Table 5.2 shows that each of the five tasks has multiple versions distinguished by their period or execution time.

Since the automated flight control system is applied in a military battle field, where security requirements such as data confidentiality are mandatory, we synthetically choose

security-required data sizes during the execution of each version of the five tasks. Each version of a task requires confidentiality, integrity, and authentication services. The security overhead for each task instance, which is computed using Equation 4.7, largely depends on the security-required data size. To evaluate the performance of our SAEDF algorithm under various scenarios, we constructed three configurations of the security-required data size for each version of the tasks (See columns 4, 5 and 6 in Table 5.2). In config1, we choose a relatively low security level for each task instance. Config2 represents a medium security level, whereas config3 reflects a relatively high security level to each task version. The experimental parameters for the automated light control system are summarized in Table 5.3.

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

We simulated a 128-node system where each node has a 1000 MIPS CPU. We considered two node-to-aircraft configurations: (1) 128 nodes were used to control 128

aircrafts, and (2) 128 nodes controlled 64 aircrafts. In the first experimental setting, each node was responsible for controlling one aircraft; and in the second setting two nodes controlled one aircraft in a cooperative manner. The goal of the first case is to test the performance under a heavily loaded condition, whereas the second case is focused on the system performance under relatively light workloads.

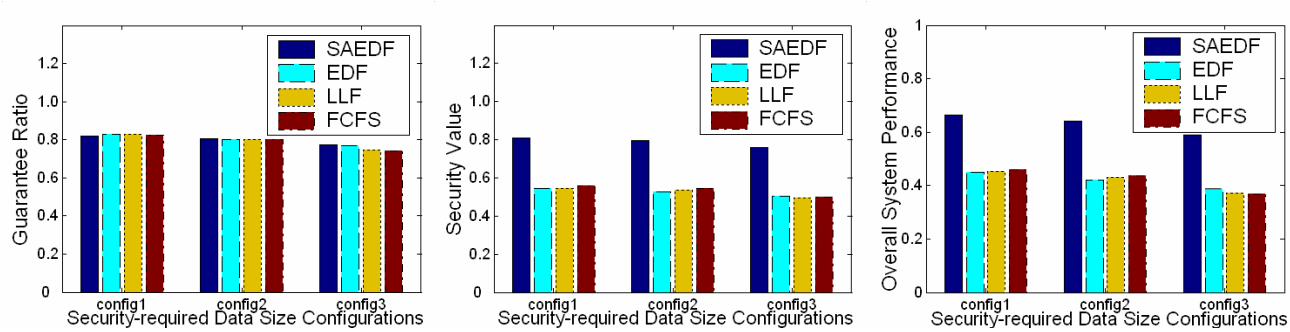
Table 5.3 Experimental Parameters for Automated Flight Control System

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

The experimental results are shown in Figure 5.14 and Figure 5.15. Three observations are evident from Figure 5.14. First, when the size of security-required data increases, guarantee ratio, security, and overall performance of the four algorithms noticeably decrease. This is because the security overhead rises with the increasing security-required data size. Consequently, the number of feasible tasks reduces, and the quality of security suffers.

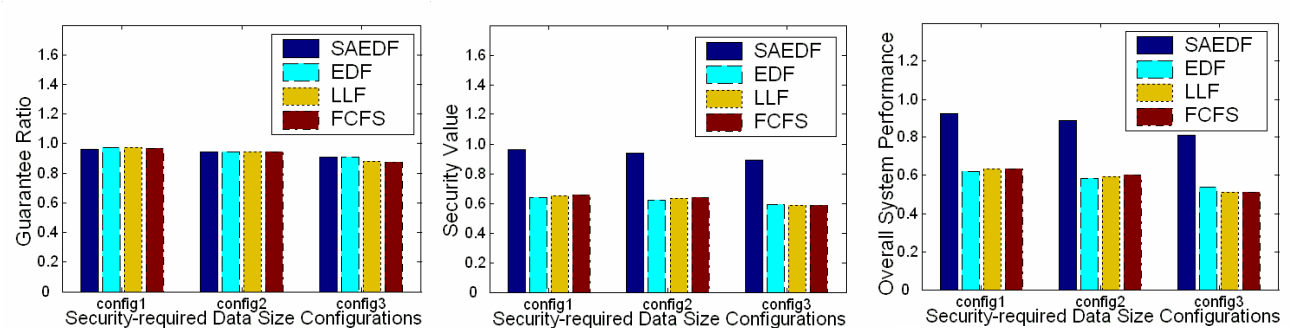
Second, when 128 nodes are utilized to control 128 aircrafts, the guarantee ratio of SAEDF on average is only 79.84%, meaning that the system is slightly overloaded. In such a workload situation SAEDF consistently outperforms the other three algorithms in quality of security (see Figure 5.14b) while maintaining the same guarantee ratio performance as EDF (see Figure 5.14a). The results demonstratively show that SAEDF

can maintain the same level of schedulability as EDF while significantly improving the security (by an average of 50.13%). More importantly, SAEDF significantly outperformed EDF, LLF and FCFS in overall system performance, which is the most crucial metric for security-critical real-time systems, by averages of 50.11%, 50.97% and 49.61%, respectively. The implication of this finding is that when system workloads are high, SAEDF can significantly improve overall system performance without adding extra hardware.



(a) (b) (c)

Figure 5.14 128 nodes control 128 F-16 aircrafts



(a) (b) (c)

Figure 5.15 128 nodes control 64 F-16 aircrafts

In the case where 128 nodes control 64 aircrafts, the average guarantee ratio of SAEDF is 93.93% (See Figure 5.15). Under such a light workload condition, the

guarantee ratios of SAEDF and the other three alternatives are almost identical. We attribute this result to the fact that when an aircraft is controlled by two computers, almost all the tasks dedicated to the aircraft can be accomplished before their deadlines because of the sufficient computational resources. In addition, the results presented in Figure 5.15 indicate that although EDF and LLF can achieve 94.12% and 93.26% in guarantee ratio, their average security values are as low as 0.62. These results suggest that EDF and LLF are unsuitable for security sensitive applications. By using SAEDF as a security-aware scheduling heuristic, performance in security value is improved by an average of 50% over EDF and LLF.

5.5 Summary

We presented in this chapter a novel security-aware heuristic strategy (SAREC) for real-time applications on clusters. This strategy paves a way to the design of security-aware real-time scheduling algorithms. The effectiveness of the SAREC strategy was evaluated by implementing a novel security-aware real-time scheduling algorithm (SAEDF), which incorporates the earliest deadline first (EDF) scheduling algorithm into the SAREC strategy. SAREC is a general strategy in the sense that it can be applied to other existing real-time scheduling policies like LLF (see Section 5.4.8).

To quantitatively validate the performance of the SAEDF algorithm, we conducted extensive trace-driven simulations and introduced two new performance metrics, namely, *security value* (see Equation 5.4) and *overall system performance* (see Equation 5.9). Security value is a collective value of each accepted application's security level and it can be used to measure the quality of security experienced by all schedulable real-time tasks. Overall system performance, the most important performance metric for security-critical

real-time systems, is a comprehensive metric defined as a product of security value and guarantee ratio. Experimental results based on real-world traces and a real application show that SAEDF achieves overall system performance over three existing scheduling algorithms (EDF, LLF, and FCFS) by average of 32.9%, 575.7%, and 713.6%, respectively. In addition, the empirical results reveal that SAEDF significantly improves quality of security for real-time tasks while maintaining high guarantee ratios under a wide range of workload characteristics.

In the next chapter we will extend the SAREC strategy to handle parallel applications, where precedence constraints must be factored in the process of security-aware scheduling. Additionally, we will extend SAREC strategy to heterogeneous cluster computing systems (see Chapter 7). In a heterogeneous cluster, different nodes have different powers and resources. Thus, the same security requirement for a particular security service will result in different amount of overheads. A node-dependable security overhead calculating model should be developed.

Chapter 6

6 A Task Allocation Scheme for Parallel Applications with Deadline and Security Constraints on Clusters

In last chapter, we address security-aware scheduling issue for independent tasks running on homogeneous clusters. To extend security-awareness in real-time scheduling for parallel applications, in this chapter we propose a new task allocation scheme, or TAPADS (Task Allocation for Parallel Applications with Deadline and Security Constraints), to find an optimal allocation that maximizes quality of security and the probability of meeting deadlines for parallel applications running on homogeneous clusters.

The chapter is organized as follows. Section 6.1 motivates this research by describing security needs for parallel application running on clusters. Section 6.2 describes mathematical models for system and parallel applications. The TAPADS algorithm is detailed in Section 6.3. Section 6.4 evaluates timeliness and security

correctness. Section 6.5 discusses the experimental results by comparing the performance of TAPADS against three existing algorithms. Section 6.6 summarizes the chapter.

6.1 Motivation

As parallel applications in most cases require massive computing power, it becomes extremely important to take advantage of cluster computing platforms where nodes are interconnected through high-speed networks, e.g. Myrinet or fast Ethernet, to meet the needs of highly parallel applications [7][66].

There have been extensive studies in the literature on scheduling of real-time parallel applications on clusters [39][64][65]. Applications with timing constraints depend not only on results of computation, but also on time instants at which these results become available [35]. Examples of real-time parallel applications include aircraft control, radar for tracking missiles, medical electronics, and on-line transaction processing systems.

There is some work addressing the issue of applications with both deadline and security constraints [75]. In particular, sensitive data and processing in various real-time applications on clusters require special safeguard and protection against unauthorized access. However, conventional task allocation schemes designed for real-time systems are inappropriate in cases where parallel applications have both real-time and security requirements. This is because the traditional task allocation schemes were merely devised to guarantee timing constraints while possibly posing unacceptable security risks.

In this chapter, we address the issue of allocating tasks of parallel applications on clusters subject to timing and security constraints in addition to precedence relationships. The proposed TAPADS scheme factors in security and timing correctness in a way that probabilities of being risk free and meeting deadlines are used as performance objectives.

In addition, we proposed mathematical models to describe a system framework, and parallel applications with deadline and security constraints.

6.2 Mathematical Models

We describe in this section mathematical models, which were built to represent a task allocation framework and parallel applications with security and deadline constraints.

6.2.1 System Model

A cluster in the most general form consists of a set $M = \{M_1, M_2, ..., M_m\}$ of nodes connected by a network. It is assumed that all nodes have an identical processing power. This assumption is reasonable in the sense that it can be easily relaxed by incorporating a simple conversion mechanism for relative heterogeneous processing capabilities. Each node communicates with other nodes through message passing, and the communication time between two tasks assigned to the same node is assumed to be negligible.

Note that the communication subsystem support messages with time constraints, meaning that the worst-case link delay is predictable and bounded. Examples of such real-time communication subsystems can be found in the literature [107]. Additionally, the communication subsystem considered in our study provides full connectivity in a way that any two nodes are connected through either a physical link or a virtual link. This assumption is arguably reasonable for modern interconnection networks (e.g. Myrinet [13]) that are widely used in high-performance clusters. Myrinet networks provide pseudo-full connectivity, allowing simultaneous transfers between any pair of nodes.

6.2.2 Task Model

6.2.2.1 Deadline and Precedence Constraints

Applications with dependent real-time tasks can be modelled by *Directed Acyclic Graphs (DAGs)*[64]. Throughout this chpater, a parallel application is defined as a vector *(T, E, p)*, where $T = \{t_1, t_2,...,t_n\}$ represents a set of non-preemptable real-time tasks, E is a set of weighted and directed edges used to represent communication among tasks, e.g., $(t_i, t_j) \in E$ is a message transmitted from task t_i to t_j, and p_i is the period. Precedence constraints of the parallel application is represented by all the edges in E. Communication time for sending a message $(t_i, t_j) \in E$ from task t_i on m_k to task t_j on m_a is determined by e_{ij}/b_{ka} where e_{ij} is the data size and b_{ka} is the network bandwidth between m_k and m_a. Message startup overhead was ignored because it is non-dominant factor compared with message transmission time.

This chapter is focused on the issue of allocating periodic tasks that are invoked at fixed time intervals and constitutes the base load of a cluster. A task is characterized by three parameters, e.g., $t_i = (e_i, l_i, S_i)$, where e_i is the execution, l_i denotes the amount of data (measured in KB) to be protected, and S_i is a vector of security requirements (see section 3.2.2.). Note that e_i can be estimated by code profiling and statistical prediction [16].

A parallel application running on the cluster generates a sequence of application instances $J_i^0, J_i^1, J_i^2 ...,$ where J_i^j must be finished before J_i^{j+1} can start executing. Note that there is a constant interval between two consecutive application instances. The deadline of J_i^j is the arrival time of the next task instance. Although the arrival time of a task instance is not explicitly specified in the model, the arrival time can be determined

when the task instance is released dynamically during the system execution. Specifically, if the initial release time of task J_i is r, the arrival time of the jth instance is $r + j \times p_i$, and the task instance has to be completed before $r + (j+1) \times p_i$. An application meets its deadline if all its instances meet their deadlines. A parallel application has a feasible schedule if for all $t \in T$, the deadline and security constraints are satisfied.

It has been proved that there exists a feasible schedule for a set of periodic tasks if and only if there is a feasible schedule for the *planning cycle* of the tasks [40]. Note that the planning cycle is the least common multiple of all the tasks' periods. Thus, the behaviour of the set of periodic tasks can be effectively analyzed within the planning cycle.

6.2.2.2 Security Constraints

A collection of security services required by task t_i is specified as $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, where S_i^j represents the required security level range of the jth security service. The TAPADS allocation scheme aims at determining the most appropriate point s_i in space S_i, e.g., $s_i = (s_i^1, s_i^2, \ldots, s_i^q)$, where $s_i^j \in S_i^j$, $1 \le j \le q$.

The proposed TAPADS scheme measures security benefits gained by parallel applications. To implement this basic functionality, we quantitatively model the security benefit of the jth task in T as a security level function denoted by $SL: S_i \rightarrow \Re$, where $\Re$ is the set of positive real numbers:

$$SL(s_i) = \sum_{k=1}^{q} w_i^k s_i^k \text{ , where } s_i = \left(s_i^1, s_i^2, ..., s_i^q\right), 0 \le w_i^j \le 1 \text{, and } \sum_{j=1}^{q} w_i^j = 1. \qquad (6.1)$$

Users can specify the weight w_i^j to reflect relative priorities given to the jth security

service. The security benefit of a task set is computed as the summation of the security levels of all the tasks. Thus,

$$SL(T) = \sum_{i=1}^{n} SL(s_i) \text{ , where } s_i = \left(s_i^1, s_i^2, ..., s_i^q\right). \tag{6.2}$$

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

$$\text{maximize } SL(T) = \sum_{i=1}^{n}\sum_{k=1}^{q} w_i^k s_i^k \text{ ,}$$

$$\text{subject to } \min(S_i^k) \le s_i^k \le \max(S_i^k), \tag{6.3}$$

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

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

$$SL(\hat{s}_{ij}) = \sum_{k=1}^{p} \hat{w}_{ij}^k \hat{s}_{ij}^k \text{ , where } \hat{s}_{ij} = (\hat{s}_{ij}^1, \hat{s}_{ij}^2, ..., \hat{s}_{ij}^p), 0 \le \hat{w}_{ij}^k \le 1, \text{ and } \sum_{j=1}^{p} \hat{w}_{ij}^k = 1. \tag{6.4}$$

Weight $\hat{w}_{ij}^k$ reflects the priority of the *k*th required security service for the message. The security benefit of a message set is calculated as the sum of the security levels of all the messages. That is,

$$SL(E) = \sum_{(t_i, t_j) \in E} SL(\hat{s}_{ij}) \text{ , where } \hat{s}_{ij} = (\hat{s}_{ij}^1, \hat{s}_{ij}^2, ..., \hat{s}_{ij}^p). \tag{6.5}$$

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

$$\text{maximize } SL(E) = \sum_{(t_i,t_j)\in E} \sum_{k=1}^{p} \hat{w}_{ij}^k \hat{s}_{ij}^k,$$

$$\text{subject to } \min(\hat{S}_{ij}^k) \le \hat{s}_{ij}^k \le \max(\hat{S}_{ij}^k), \tag{6.6}$$

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

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

$$\text{maximize } SV = SL(T) + SL(E). \tag{6.7}$$

Substituting Equations 6.3 and 6.6 into 6.7 yields the following security value objective function. The task allocation problem can be formally defined as follows: given a cluster M = *{M_1, M_2, …, M_n}* and a parallel application *{T, E}*, find an allocation of each task and message, such that the following total security level of the application on the cluster is maximized.

$$SV = \sum_{i=1}^{n} \sum_{k=1}^{q} w_i^k s_i^k + \sum_{(t_i,t_j)\in E} \sum_{k=1}^{p} \hat{w}_{ij}^k \hat{s}_{ij}^k, \tag{6.8}$$

6.3 The Task Allocation Scheme TAPADS

This section presents a task allocation algorithm (TAPADS) for parallel applications with deadline and security constraints on clusters. Let *X* be an *m* by *n* binary matrix corresponding to an allocation, in which *n* tasks are assigned to *m* nodes in the cluster. Element x_{ij} equals 1 if and only if t_i has been allocated to node m_j; otherwise $x_{ij} = 0$. Thus, we have $M(t_i) = j \Leftrightarrow x_{ij} = 1$, where *M($t_i$)* indicates the node to which t_i is allocated. The

TAPADS algorithm strives to maximize $P_{OSP}(X) = P_{SC}(X) \cdot P_{SD}(X)$ subject to $\sum_{j=1}^{m} x_{ij} = 1$ $i \in [1, n]$, where $P_{OSP}(X)$ is the overall system performance, $P_{SC}(X)$ is the probability that all tasks are executed without any risk of being attacked and all messages are risk-free during the course of message transmissions, and $P_{SD}(X)$ is the probability that all tasks are completed on time. Note that $P_{SC}(X)$ and $P_{SD}(X)$ will be derived in Section 6.4.

1. Sort and renumber tasks so that if $(t_i, t_j) \in E$ then $i < j$;
2. Compute the critical path of the task graph;
3. Calculate the tentative finish time, $f = \sum_{t_i \in \text{critical path}} (e_i + c_i^{\min})$;
4. Allocate and schedule all t_i in the critical path subject to minimal security requirements;
5. Allocate and schedule all $t_i \notin$ critical path subject to precedence and minimal security constraints;
6. Obtain slack time, $slk = d - f$, where d is the deadline;
7. **while** (slack time ≥ 0) **do**
   - 7.1 select *i'* and *j'* subject to
     $$w_{i'}^{j'} \Delta s_{i'}^{j'} \Big/ \left(c_{i'}^{j'}(s_{i'}^{j'} + \Delta s_{i'}^{j'}) - c_{i'}^{j'}(s_{i'}^{j'})\right) = \max_{1 \le i \le n, 1 \le j \le q} \left\{ w_i^j \Delta s_i^j \Big/ \left(c_i^j(s_i^j + \Delta s_i^j) - c_i^j(s_i^j)\right) \right\}$$
   - 7.2 select *a'* and *b'* subject to
     $$\hat{w}_{a'b'}^{k'} \Delta \hat{s}_{a'b'}^{k'} \Big/ \left(\hat{c}_{a'b'}^{k'}(\hat{s}_{a'b'}^{k'} + \Delta \hat{s}_{a'b'}^{k'}) - \hat{c}_{a'b'}^{k'}(\hat{s}_{a'b'}^{k'})\right) = \max_{\substack{(t_a, t_b) \in E \\ 1 \le k \le p}} \left\{ \hat{w}_{ab}^{k} \Delta \hat{s}_{ab}^{k} \Big/ \left(\hat{c}_{ab}^{k}(\hat{s}_{ab}^{k} + \Delta \hat{s}_{ab}^{k}) - \hat{c}_{ab}^{k}(\hat{s}_{ab}^{k})\right) \right\}$$
   - 7.3 **if** $w_{i'}^{j'} \Delta s_{i'}^{j'} \Big/ \left(c_{i'}^{j'}(s_{i'}^{j'} + \Delta s_{i'}^{j'}) - c_{i'}^{j'}(s_{i'}^{j'})\right) > \hat{w}_{a'b'}^{k'} \Delta \hat{s}_{a'b'}^{k'} \Big/ \left(\hat{c}_{a'b'}^{k'}(\hat{s}_{a'b'}^{k'} + \Delta \hat{s}_{a'b'}^{k'}) - \hat{c}_{a'b'}^{k'}(\hat{s}_{a'b'}^{k'})\right)$ **then**
     increase security level $s_{i'}^{j'} \leftarrow s_{i'}^{j'} + \Delta s_{i'}^{j'}$ if $s_{i'}^{j'} < \max\{S_{i'}^{j'_l}\}$;
   - 7.4 **else** increase security level $\hat{s}_{a'b'}^{k'} \leftarrow \hat{s}_{a'b'}^{k'} + \Delta \hat{s}_{a'b'}^{k'}$ if $\hat{s}_{a'b'}^{k'} < \max\{\hat{S}_{a'b'}^{k'_l}\}$;
   - 7.5 update task schedule, e.g., start times of tasks and messages;
   - 7.6 update slack time based on the increased security level;

   end while

Figure 6.1 The TAPADS task allocation algorithm

The TAPADS algorithm is outlined in Figure 6.1. TAPADS aims at achieving high quality of security under two conditions: (1) increasing security levels will not result in missing deadlines; and (2) precedence constraints are satisfied. In an effort to meet both deadline and precedence constraints, TAPADS assigns the tasks to each node in a way to

maximize security measured as $P_{SC}(X)$. Thus, TAPADS is capable of maintaining a high schedulability measured as $P_{SD}(X)$.

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

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

$$\theta_i^j = w_i^j \Delta s_i^j / \left(c_i^j (s_i^j + \Delta s_i^j) - c_i^j (s_i^j) \right), \text{for the } j\text{th service of task } i, \tag{6.9}$$

$$\hat{\theta}_{ij}^k = \hat{w}_{ij}^k \Delta \hat{s}_{ij}^k / \left(\hat{c}_{ij}^k (\hat{s}_{ij}^k + \Delta \hat{s}_{ij}^k) - \hat{c}_{ij}^k (\hat{s}_{ij}^k) \right), \text{for the } k\text{th service of message } (t_i, t_j), \tag{6.10}$$

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

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

$$\begin{cases} \theta_{i'}^{j'} = \max\limits_{1\le i\le n, 1\le j\le q}\left\{\theta_i^j\right\}, \text{if } \max\limits_{1\le i\le n, 1\le j\le q}\left\{\theta_i^j\right\} \ge \max\limits_{(t_i,t_j)\in E, 1\le k\le p}\left\{\hat{\theta}_{ij}^k\right\}, \\ \theta_{i'j'}^{k'} = \max\left\{\max\limits_{(t_i,t_j)\in E, 1\le k\le p}(\hat{\theta}_{ij}^k)\right\}, \qquad \text{otherwise.} \end{cases} \tag{6.11}$$

To yield a maximized security level of the job, Steps 7.3 and 7.4 are responsible for increasing security levels of more important services at the minimal cost. Thus, the slack time is distributed on a task or message with the highest benefit-cost ratio. Step 7.5 updates the schedule in accordance with the increased security level, because start times of other tasks and messages are dependent of how the slack time is distributed. The time complexity of the *SAREG* scheduling algorithm is given as follows.

Theorem 6.1. The time complexity of TAPADS is $O(k(q|V|+p|E|))$, where k is the number of times Step 7 is repeated, q is the number of security services for computation, p is the number of security service for communication.

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

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

6.4 Evaluation of Timeliness and Security Correctness

In this section, we first explain a way of scheduling tasks allocated to a node. Then, we derive the probability $P_{SD}(X)$ that all tasks meet the deadline constraints. Finally, we calculate the probability $P_{SC}(X)$ that all tasks and messages are risk-free during the execution of the job. Note that $P_{SD}(X)$ and $P_{SC}(X)$ help in evaluating the performance of our task allocation scheme in Section 6.5.

6.4.1 Task and Message Scheduling

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

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

$$eat_k(t_j, t_i) = \begin{cases} f(t_j) & \text{if } M(t_i) = M(t_j) \\ mst_{uv}(t_j, t_i) + d_{ji} / B_{uv} & \text{otherwise} \end{cases}. \qquad (6.12)$$

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

$$eat_k(t_i) = \max_{(t_j,t_i)\in E}\{eat_k(t_j, t_i)\}. \tag{6.13}$$

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

$$est(t_i) = \min_{M_j\in M}\{est_j(t_i)\}. \tag{6.14}$$

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

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

$$\forall 1 \le i \le n : est(t_i) + e_i + c_i^{min} \le d\ , \tag{6.15}$$

where $est(t_i)$ is the earliest start time obtained from expression (6.15), d is the deadline of the job, e_i is the computation time, and c_i^{min} is the security overhead experienced by task t_i when its minimal security requirements are met.

The minimal security overhead c_i^{min} can be calculated by the following equation, where $c_i^j\left(\min\left\{S_i^j\right\}\right)$ is the overhead of the jth security service when the minimal security

requirement is met.

$$c_i^{min} = \sum_{j=1}^{q} c_i^j \left(\min\left\{S_i^j\right\}\right). \quad (6.16)$$

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

$$\forall 1 \le i \le n : est(t_i) + e_i + \sum_{j=1}^{q} c_i^j (s_i^j) \le d. \quad (6.17)$$

The allocation X is feasible if all tasks can be completed before the deadline. Therefore, the probability

$P_{SD}(\text{tasks are timely completed under } X \text{ with security level vector } \bar{s})$ is expressed as below, where $\bar{s} = (s_1, s_2, ..., s_n)$, $\varphi_i(s_i) = 1$ for $est(t_i) + e_i + \sum_{j=1}^{q} c_i^j (s_i^j) \le d$, and $\varphi_i(s_i) = 0$ otherwise.

$$P(\text{tasks are completed on time under } X \text{ with security level vector } \bar{s}) = \prod_{i=1}^{n} \varphi_i(s_i), \quad (6.18)$$

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

$$P_{SD}(X) = \sum_{all\ \bar{s}_k} p_k P(\text{tasks are completed on time under } X \text{ with security level vector } \bar{s}_k)$$

$$= \sum_{all\ \bar{s}_k} p_k \prod_{i=1}^{n} \varphi_i(s_{ki}). \quad (6.19)$$

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

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

To evaluate quality of security for parallel applications, we derive in this section the probability $P_{SC}(X)$ that all tasks and messages remain risk-free during the course of execution. The quality of security of a task t_i with respect to the *j*th security service is calculated as $\exp(-\lambda_i^j e_i)$, where λ_i^j is t_i's risk rate of the *j*th security service and e_i is the execution time. The risk rate is expressed as:

$$\lambda_i^j = 1 - \exp\left(-\alpha(1 - s_i^j)\right). \tag{6.20}$$

Note that this risk rate model assumes that risk rates are a function of security levels, and the distribution of risk-count for any fixed time interval is approximated using a *Poisson* probability distribution. This 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 a task t_i can be obtained below by considering all security services provided to the task. Consequently, we have:

$$\prod_{j=1}^{q} \exp(-\lambda_i^j e_i) = \exp\left(-e_i \sum_{j=1}^{q} \lambda_i^j\right). \tag{6.21}$$

Given an allocation *X*, the probability that all tasks are free from being attacked during the execution of the tasks is computed based on Equation (6.21). Thus,

$$P_C(X) = \prod_{i=1}^{n} \exp\left(-e_i \sum_{j=1}^{q} \lambda_i^j\right). \tag{6.22}$$

By substituting the risk rate model into Equation (6.22), we finally obtain $P_C(X)$ as shown below:

$$P_C(X) = \prod_{i=1}^{n} \exp\left\{ -e_i \sum_{j}^{q} \left[1 - \exp\left(-\alpha (1 - s_i^j) \right) \right] \right\}. \qquad (6.23)$$

Likewise, for the *k*th security service available for a link between M_i and M_j, the quality of security of the link during the time interval *t* is $\exp(-\hat{\lambda}_{ij}^k t)$, where $\hat{\lambda}_{ij}^k$ denotes the risk rate of the *k*th security service. The risk rate is expressed as the following function of the corresponding security level.

$$\hat{\lambda}_{ij}^k = 1 - \exp\left(-\beta (1 - s_{ij}^k) \right). \qquad (6.24)$$

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

$$\prod_{k=1}^{p} \exp(-\hat{\lambda}_{ij}^k \frac{d_{ab}}{B_{ij}}) = \exp\left(-\frac{d_{ab}}{B_{ij}} \sum_{k=1}^{p} \hat{\lambda}_{ij}^k \right), \text{ where } x_{ai} = 1, x_{bj} = 1. \qquad (6.25)$$

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

$$P_{ij}(X) = \prod_{a=1}^{n} \prod_{b=1, b \neq a}^{n} \exp\left[x_{ai} x_{bj} \left(-\frac{d_{ab}}{B_{ij}} \sum_{k=1}^{p} \hat{\lambda}_{ij}^k \right) \right]. \qquad (6.26)$$

We now obtain $P_{ij}(X)$ by substituting the risk rate model into Equation (6.26),

$$P_{ij}(X) = \prod_{a=1}^{n} \prod_{b=1, b \neq a}^{n} \exp\left\{ x_{ai} x_{bj} \left[-\frac{d_{ab}}{B_{ij}} \sum_{k=1}^{p} \left(1 - \exp\left(-\beta (1 - s_{ij}^k) \right) \right) \right] \right\}. \qquad (6.27)$$

Let $P_L(X)$ be the quality of security experienced by all links under allocation *X*, then $P_L(X)$ can be written as the following equation:

$$P_L(X) = \prod_{i=1}^{m} \prod_{j=1, j\neq i}^{m} P_{ij}(X). \quad (6.28)$$

Finally, the probability $P_{SC}(X)$ can be calculated as follows, where $P_C(X)$ and $P_L(X)$ are obtained from Equations (6.23) and (6.28).

$$P_{SC}(X) = P_C(X)P_L(X). \quad (6.29)$$

6.5 Performance Evaluation

Now we are in a position to evaluate the effectiveness of TAPADS using extensive simulations and a real application case study. To demonstrate the strength of TAPADS, we compare it with list scheduling scheme, which is a well-known scheduler for parallel applications. To make the comparison fair, we slightly modified it into three variants: LISTMIN, LISTMAX, and LISTRND. The variants can meet parallel applications' security requirements in a heuristic way. Although these algorithms are intended to schedule real-time tasks with security requirements, they make no effort to optimize quality of security. The three baseline algorithms are briefly described below.

(1) *LISTMIN*: The scheduler intentionally selects the lowest security level of each security service required by each task of a parallel job. Thus, given task T_i and service v_l, the following equation always holds: $s_i^{v_l} = \min\{S_i^{v_l}\}$.

(2) *LISTMAX*: The scheduler chooses the highest security level for each security requirement posed by each task within a parallel job. This fact can be formally represented by the following expression: $\forall T_i, v_l \in \{a, e, g\} : s_i^{v_l} = \max\{S_i^{v_l}\}$.

(3) *LISTRND*: Unlike the above two baseline algorithms, LISTRND randomly picks a value within the security level range of each service required by a task. The following expression is held in LISTRND: $\forall T_i, v_l \in \{a, e, g\} : s_i^{v_l} = \text{random}\{S_i^{v_l}\}$.

6.5.1 Simulator and Simulation Parameters

Before presenting empirical results, we present the simulation model as follows. Table 6.1 summarizes the configuration parameters of simulated clusters used in our experiments. The parameters of nodes in the clusters are chosen to resemble real-world workstations like Sun SPARC-20 and Sun Ultra 10.

Table 6.1 Characteristics of System Parameters

Parameter	Value (Fixed) - (Varied)
CPU Speed	100 million instructions/second or MIPS
Network bandwidth	5 MB/second
Task execution time	(min, top, max)=(1, 5, 10), (10,20,40), (40,80,160), (160,320,640) second
Number of nodes	(32, 64,128, 256), (8, 12, 16, 20)
Deadlines	(100, 200, 300, 400, 500, 600) second
Deadline ranges	([100, 200], [200, 300], [300, 400], [400, 500]) second
Out degrees	(25, 50, 75, 100)
Size of data to be secured	(min, top, max)=(0.02, 0.1, 0.5), (0.2, 1, 5), (1, 5, 10), (10, 20, 30) MB
Required security services	Encryption, Integrity, and Authentication
Weight of security services	0.2 (authentication), 0.5 (encryption), 0.3 (integrity)

All synthetic parallel jobs used from Section 6.5.2 to Section 6.5.7 were created by TGFF [24], a randomized task graph generator. The task graph used in Section 6.5.8 is based on a digital signal processing system detailed in [93].

Although deadlines, size of data to be secured, numbers of out degree, task execution time are synthetically generated, we examined impacts of these important workload

parameters on system performance by controlling the parameters. The performance metrics by which we evaluate system performance include:

- *Security Value* (see Equation 6.8).
- *Schedulability*: a fraction of total submitted jobs that are schedulable (see Equation 6.19).
- *Quality of security (QSA)*: quality of security for applications (see Equation 6.29).
- *Guarantee factor*: it is zero if a job's deadline cannot be met. Otherwise, it is one.

Job completion time: earliest time that a job can finish its execution.

6.5.2 Overall Performance Comparisons

The goal of this experiment is to compare the proposed *TAPADS* algorithm against the other three baseline schemes, and to understand the sensitivity of *TAPADS* to deadlines. We tested task graphs with 433 tasks with precedence constraints.

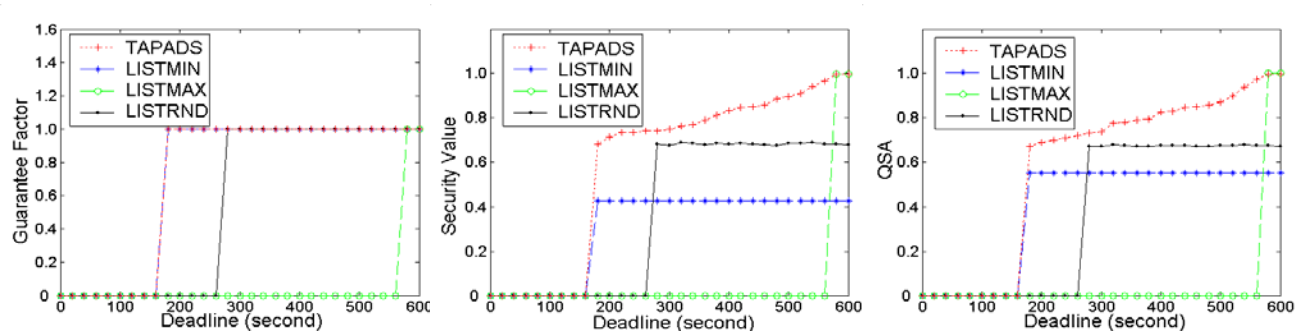


Figure 6.2 Performance impact of deadline

Figure 6.2 shows the simulation results for these four algorithms on a cluster with 32 nodes. We statically computed the minimal job completion time defined as a job's completion time when the minimal security requirements of each task and message are

met. Similarly, we can calculate the random job completion time and the maximal job completion time. For the tested parallel applications, the minimal, random, and maximal job completion times are 170 second, 260 second and 575 second, respectively. We observe from Figure 6.2a that TAPADS and LISTMIN exhibit the same performance in terms of guarantee factor, whereas TAPADS noticeably outperforms the LISTMAX and LISTRND algorithms. Once deadline is longer than 170 seconds, both TAPADS and LISTMIN have the capability of generating feasible schedules, whereas no feasible schedule can be yielded by LISTRND and LISTMAX until deadlines are longer than 260 and 575 seconds, respectively. This is because high security overhead results in long completion times. We attribute the performance improvement of TAPADS over LISTMAX and LISTRND to the fact that TAPADS boosts the security levels of tasks and messages under the condition that timing constraints are guaranteed and, therefore, TAPADS maintains the same guarantee factor performance as that of LISTMIN. In contrast, the LISTMAX and LISTRND policies improve the quality of security at the cost of missing deadlines.

Figure 6.2b plots security values of the four algorithms when the deadline is increased from 0 to 600 seconds. It reveals that TAPADS consistently performs better than LISTMIN and LISTRND. In particular, TAPADS achieves improvement on averages of 97.7% and 25%, respectively. This is because LISTMIN and LISTRND do not utilize slack times to increase security levels of tasks and messages. In the case where the deadlines are longer than the maximal job completion time, TAPADS eventually degrades to LISTMAX. Importantly, TAPADS substantially outperforms LISTMAX when the deadlines are tight. The results clearly indicate that clusters can gain more

performance benefits from our TAPADS approach under the circumstance that real-time applications have relatively tight deadlines.

Improvements in the quality of security achieved by TAPADS are plotted in Figure 6.2 (c). Compared with LISTMIN and LISTRND, TAPADS achieves improvement on the average of 54.5% and 25.7%, respectively. The first observation deduced from Figure 6.2c is that the quality of security of TAPADS increases with the deadline. This is because quality of security is partially derived from SV (see Equations 6.22 and 6.26), which becomes higher when the deadlines are looser. A second observation is that the performance improvement of TAPADS in terms of quality of security is not as pronounced as the performance improvement in terms of security value compared with LISTMIN algorithm. This can be explained by the negative natural exponential function (see Equation 6.20 and Equation 6.24), which smoothes the security value differences between LISTMIN and TAPADS.

6.5.3 Adaptability

We conducted four groups of experiments to test the performance of TAPADS using 1000 diverse task graphs. The smallest task graph has 54 tasks, and the largest task graph consists of 543 tasks. We assume that the number of nodes in the cluster is 32. For each group test, we set a deadline range from which a deadline is randomly selected for an incoming parallel job. The four deadline ranges for the four group experiments are [100, 200], [200, 300], [300, 400] and [400, 500], respectively.

Figure 6.3 shows the performance impacts of the deadlines. We observe from Figure 6.3a that the TAPADS and LISTMIN deliver the best performance in schedulability under all four cases. Figure 6.3b demonstrates that TAPADS consistently outperforms the

others in terms of security value. Interestingly, the improvement of TAPADS in security over LISTMAX is significant, although LISTMAX attempts to meet the maximal security requirements. This is mainly due to the low schedulability of LISTMAX (e.g., LISTMAX merely provides feasible schedules for 50% parallel applications). Figure 6.3c clearly shows that TAPADS noticeably delivers the best quality of security.

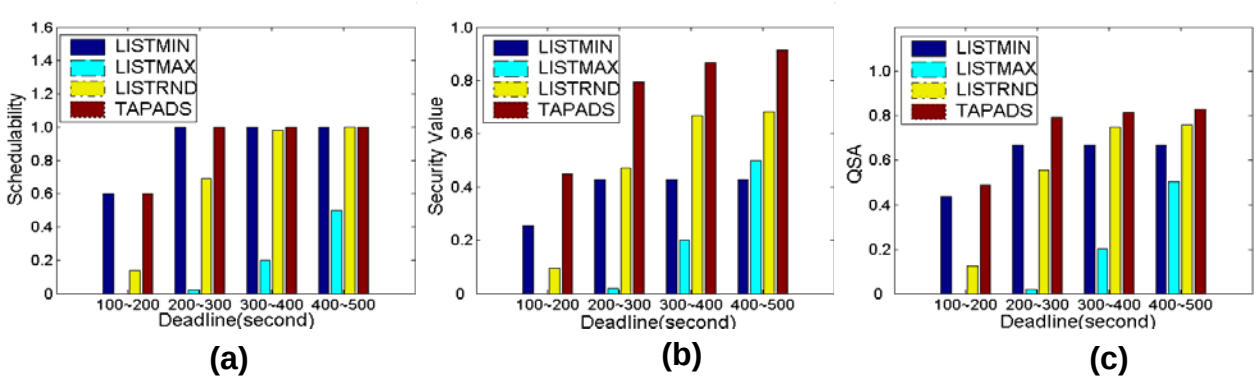


Figure 6.3 Group experiment for deadline

6.5.4 Scalability

This experiment is intended to investigate the scalability of the TAPADS algorithm. We scale the number of nodes in the cluster from 32 to 256. The task graph used in this experiment is comprised of 520 tasks, and the deadline is set to 400 Sec. Figure 6.4 plots the performance of the four algorithms as functions of the number of nodes. The results show that TAPADS exhibits the best scalability.

Figure 6.4a shows the improvement of TAPADS in security value over the other three heuristics. It is observed from Figure 6.4a that the improvement of TAPADS over LISTMIN becomes more prominent with the increasing value of the node number. This result can be explained by the conservative nature of LISTMIN, which simply meets the minimal security requirements for parallel applications on the cluster. Conversely, LISTMAX can achieve the same performance as TAPADS when there are 256 nodes in

the cluster. This is because LISTMAX can guarantee the maximal security requirements of the parallel jobs when more nodes are available in the cluster. We observe from Figure 6.4c that (1) all the four algorithms can finish the job in a shorter time period when there is large number of available nodes; (2) TAPADS has the same performance in complete time as that of LISTMIN. First observation can be explained by the fact that more independent tasks can be executed concurrently when more nodes are available. The second observation is due to the ability of TAPADS and LISTMIN to fulfill the minimal security requirements for parallel applications. It is worth noting that TAPADS substantially outperforms LISTMIN with respect to security value and quality of security.

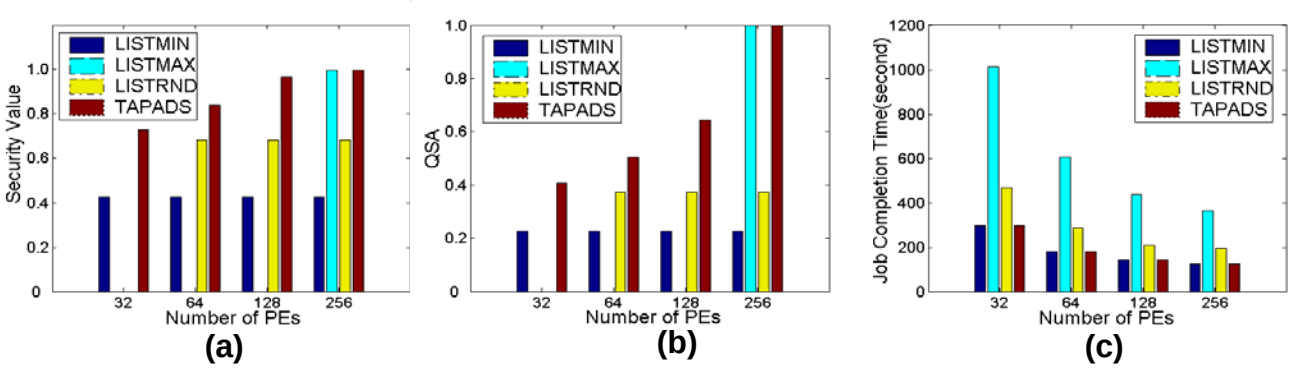


Figure 6.4 Performance impact of number of PEs

6.5.5 Sensitivity to degree of task parallelism

To verify the performance impact of the degree of task parallelism, we evaluate the performance as functions of maximal number of out degree in task graphs. We define the degree of task parallelism of a task graph as the maximal out degree in the graph. The larger the maximal out degree number is, the higher the degree of task parallelism due to the large number of independent tasks concurrently running on the cluster. We tested a parallel application with 1074 tasks. The deadline is set to 400 Sec., and the number of

nodes is 128. The maximal number of out degree varies from 25 to 100. Figure 6.5a reveals that in terms of security value TAPADS outperforms the others in the first three cases and ties with LISTMAX in the last situation.

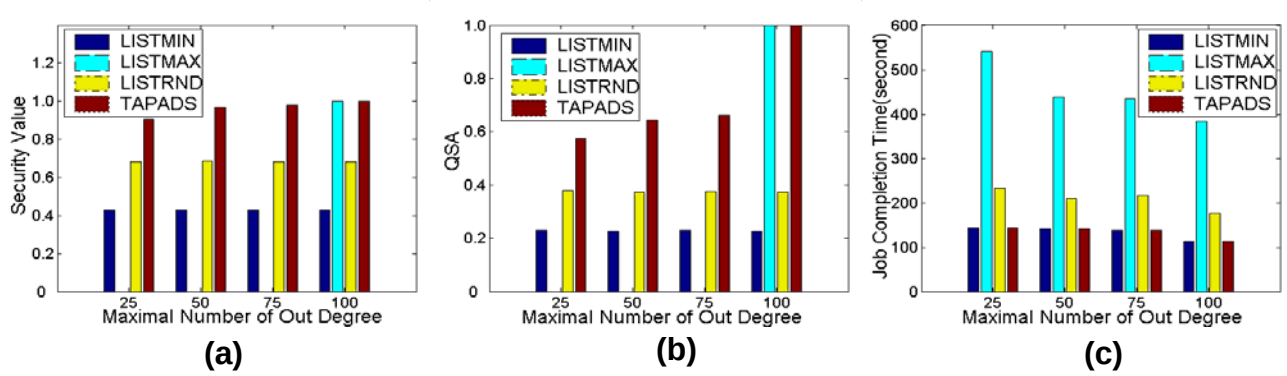


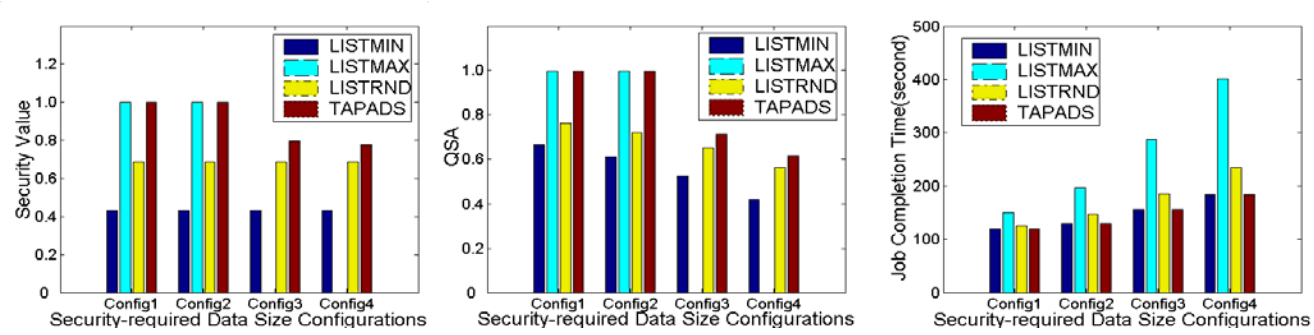
Figure 6.5 Performance impact of maximal number of out degree

Figure 6.5c demonstrates that the job completion time decreases, meaning that the increasing value of out degree leads to long slack times. This is because the high task parallelism allows many independent tasks to be simultaneously executed on multiple nodes. Interestingly, among the four alternatives TAPADS is the only algorithm that can continuously improve its performance in security value and quality of security with the increasing value of task parallelism. This phenomenon can be explained by the fact that TAPADS is conducive to leveraging slack times to increase security levels. The important conclusion drawn from this experiment is that TAPADS can gain greater performance improvement when a parallel application has a higher degree of parallelism, which reflects a salient feature of scalable parallel applications.

6.5.6 Impact of size of security sensitive data

In this experiment we evaluate the performance impact of size of security sensitive data. 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 job completion time because the larger the data, the longer time is needed to secure it. In this experiment, the size of security sensitive data varies from (0.02, 0.1, 0.5) to (10, 20, 30) MB in a triangle distribution (see Table 6.1).



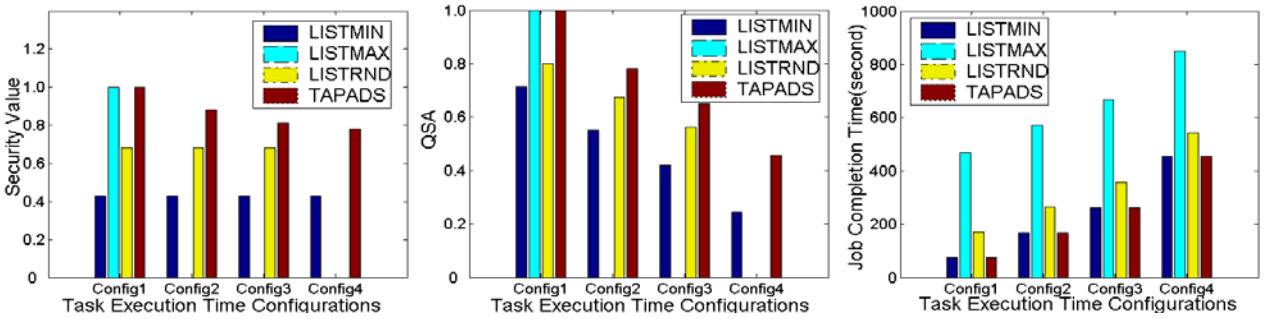
(a) (b) (c)

Figure 6.6 Performance impact of size of security-required data

The empirical results are shown in Figure 6.6. There are several important observations drawn from Figure 6.6. Firstly, when the security sensitive data size varies from config1 to config4, the security value of TAPADS drops, while those of LISTMIN and LISTRND remain the same. This is because the security overhead is increasing and thus less slack time can be used by TAPADS to improve security values. Secondly, the quality of security for LISTMIN and LISTRND decreases when the size of security sensitive data goes up, although their security values keep unchanged. This interesting phenomena can be explained by Equations 6.22 and 6.26, which indicate long execution and communication times lower QSA values. Lastly, Figure 6.6c illustrates that the increasing size of security sensitive data increases the job completion time. This is because when size of data to be secured rises, time spent in security services becomes longer.

6.5.7 Impact of task execution time

The purpose of this experiment is to show the performance impact of task execution times. Again, we simulated parallel applications with 433 tasks. The deadline is fixed at 500 seconds, and the number of nodes is set to 32. There are four task execution time configurations (see Table 6.1). Figure 6.7a shows that increasing task execution times reduces the security value of TAPADS. This result is consistent with Section 6.5.6 because long task execution times imply short slack time. Figure 6.7b exhibits the same interesting scenario as that depicted in Figure 6.6b, i.e., increasing execution times lowers QSA values. It is noticed from Figure 6.7 that LISTMAX fails in generating feasible schedules in the Configurations 2-4. This is because to provide feasible schedulers when the computation demands are high, LISTMAX requires loose deadlines.



(a) (b) (c)

Figure 6.7 Performance impact of task execution time

6.5.8 Evaluation in Real Application

To validate the results from the synthetic simulations above, we evaluate the TAPADS algorithm in a real system – digital signal processing system [93].

The detailed information pertinent to the DSP application can be found in [93]. We conducted two groups of experiments. The first group of experiments (Figure 6.8) is focused on the impact of deadline, whereas the second group (Figure 6.9) is intended to test the scalability of TAPADS in the context of a real world parallel application.

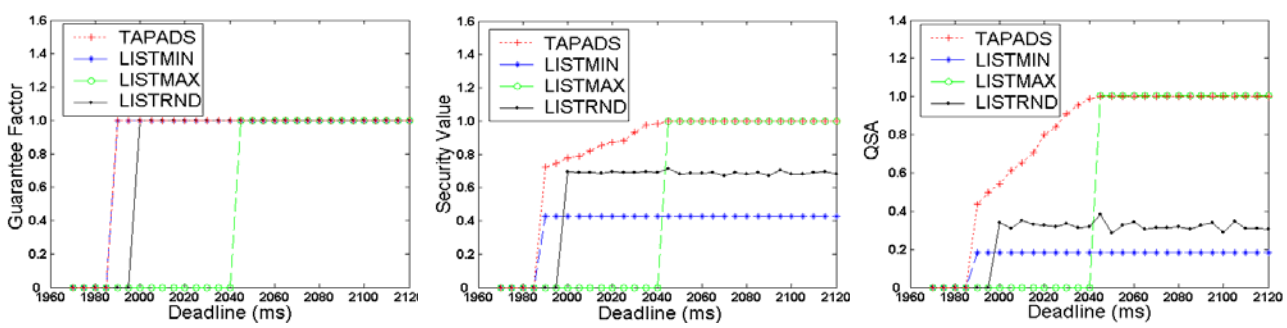


Figure 6.8 Performance impact of deadline for DSP

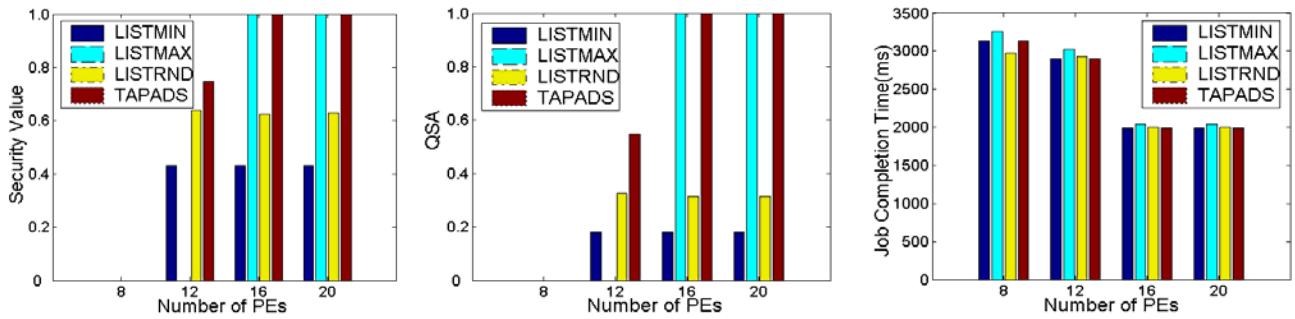


Figure 6.9 Performance impact of number of nodes for DSP

Performance patterns plotted in Figure 6.8 are similar to those reported in Section 6.6.2 (see Figure 6.2), thereby verifying that TAPADS can gain performance improvements for a real application. Figure 6.9 shows that at least 12 nodes are required to make feasible scheduling decisions for the DSP application. When the number of nodes is equal to or larger than 16, TAPADS ties with LISTMAX in terms of security value and QSA. This is because the two algorithms can produce feasible schedules under the workload where there exists sufficient number of nodes.

In summary, the strength of TAPADS can be fully exhibited when the application has a relatively tight deadline. When the deadline is extremely loose, TAPADS degrades to LISTMAX. The implication is that TAPADS can significantly improve security for

real-time applications without increasing hardware cost. The results discussed in this subsection can be envisioned as a strong validation of our previous simulations. The salient feature of TAPADS is that it can be successfully deployed to secure real-time parallel applications on clusters.

6.6 Summary

In this chapter, we address the issue of allocating tasks of parallel applications on clusters subject to timing and security constraints in addition to precedence relationships. TAPADS (Task Allocation for Parallel Applications with Deadline and Security Constraints), a task allocation scheme for parallel application with deadline and security constraints, is developed to generate optimal allocations that maximize quality of security and the probability of meeting deadlines for parallel applications running on clusters. The proposed TAPADS scheme factors in security and timing correctness in a way that the probabilities of being risk free and meeting deadlines are used as the performance objectives for clusters. To facilitate the presentation of TAPADS, we also proposed mathematical models to describe a system framework, and parallel applications with deadline and security constraints. To quantitatively evaluate the effectiveness and practicality of the TAPADS scheme, we conducted extensive experiments using a real world application as well as synthetic benchmarks. Our experimental results show that TAPADS significantly improves the performance of clusters in terms of security and schedulability over three existing allocation schemes. Compared with LISTMIN and LISTRND, TAPADS achieves improvements in security value on averages of 97.7% and 25%, respectively. The improvements of TAPADS in quality of security over LISTMIN

and LISTRND are 54.5% and 25.7%, respectively. Moreover, TAPADS improves the schedulability performance over LISTMAX by an average of 400.0%.

In Chapter 7 we will extend the TAPADS scheme to deal with heterogeneous clusters, where different nodes have various computing capacities and resources.

Chapter 7

7 A Security and Heterogeneity Driven Scheduling Algorithm

In Chapters 5 and 6, we proposed two security-aware scheduling schemes for homogeneous clusters. Although these algorithms can improve system security and performance, they have no inherent capability of supporting heterogeneous clusters. In this chapter, we present a security and heterogeneity driven scheduling algorithm (SHARP for short).

The rest of the chapter is organized as follows. In the section that follows, the motivation for the development of the SHARP algorithm is presented. Section 7.2 introduces a system model and a new concept of security heterogeneity. Section 7.3 focuses on the description of the SHARP algorithm. Section 7.4 evaluates security risks experienced by each task. Section 7.5 discusses performance measurements of the proposed scheme by simulating a heterogeneous cluster of 64 nodes. Finally, Section 7.6 summarizes the main contributions of this chapter.

7.1 Motivation

A heterogeneous cluster consists of a group of diverse computers, called nodes, which are connected by a high-performance network. To date heterogeneous clusters have been emerging as popular computing platforms for computationally intensive applications with diverse computing needs. Scheduling algorithms play a key role in obtaining high performance in parallel systems like heterogeneous clusters [53]. Scheduling algorithms fall into two camps: static [48][73] and dynamic [65][99][97]. The objective of scheduling algorithms is to map tasks onto nodes and order their execution in a way to optimize overall performance. In this work we consider the issue of dynamic scheduling for parallel applications that have subtasks.

Recently there were some efforts devoted to development of real-time parallel applications on clusters [5][65]. Real-time parallel applications depend not only on results of computation, but also on time instants at which these results become available. Today there exists a growing number of parallel applications demanding both high security and real-time performance, because sensitive data and parallel processing require special safeguards and protections against unauthorized access. Inherently, one of the challenges in heterogeneous cluster computing is to develop scheduling algorithms for real-time parallel applications with a diversity of security requirements. The reason is two-fold. First, large-scale heterogeneous clusters are vulnerable to threats as information can be accessed over networks. Second, it is difficult to control parallel processing activities with security and timing constraints. In recent years, many studies addressed the issue of both real-time and non-real-time scheduling for heterogeneous systems in general and heterogeneous clusters in particular [16][20][65][68][84]. However, existing

scheduling algorithms perform poorly for security-sensitive parallel applications with real-time requirements due to the ignorance of either security or timeliness requirements imposed by parallel applications.

Much attention has been focused on security for applications running on clusters [6] [9][95]. In addition, conventional security techniques like authentication [71] and access control [72] can be successfully deployed in clusters. However, most existing security techniques are not appropriate for real-time applications due to the lack of ability to express and handle timing constraints. Xie *et al.* proposed an array of security-aware scheduling algorithms for clusters [97] and grids [99]. Although these algorithms can improve system security and performance, their algorithms have no inherent capability of supporting heterogeneous clusters. In this regard, we are motivated to introduce the concept of security heterogeneity, and to propose a security and heterogeneity driven scheduling algorithm to improve security of real-time parallel applications on heterogeneous clusters.

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

well-know scheduling algorithms. Extensive experimental studies using real world traces confirm that the proposed SHARP algorithm significantly improves security and performance of parallel applications on heterogeneous clusters.

7.2 System Models

7.2.1 Security Driven Scheduling Architecture

In this section, we focus on a scheduling architecture for parallel applications with security and real-time constraints. As depicted in Figure 7.1, a heterogeneous cluster is comprised of n heterogeneous nodes connected via a high-performance network (e.g. Myrinet or fast Ethernet [88]) to process parallel applications submitted by users. Note that throughout this chapter the terms application and job are used interchangeably. Let $N = \{N_1, N_2, \ldots, N_n\}$ denote the set of heterogeneous nodes. The scheduling architecture consists of a real-time scheduler, an admission controller, and a security driven task allocator. The admission controller is deployed to perform feasibility checks by determining if an arriving parallel application can be completed by the heterogeneous cluster before the specified deadline. The parallel application will be admitted into the cluster if the application's deadline can be met by the cluster. The real-time scheduler is to satisfy timing requirements of parallel applications by assigning high priorities to applications with early deadlines. The security driven task allocator is intended to generate task allocation decision for each task of a parallel application, satisfying both security and real-time requirements.

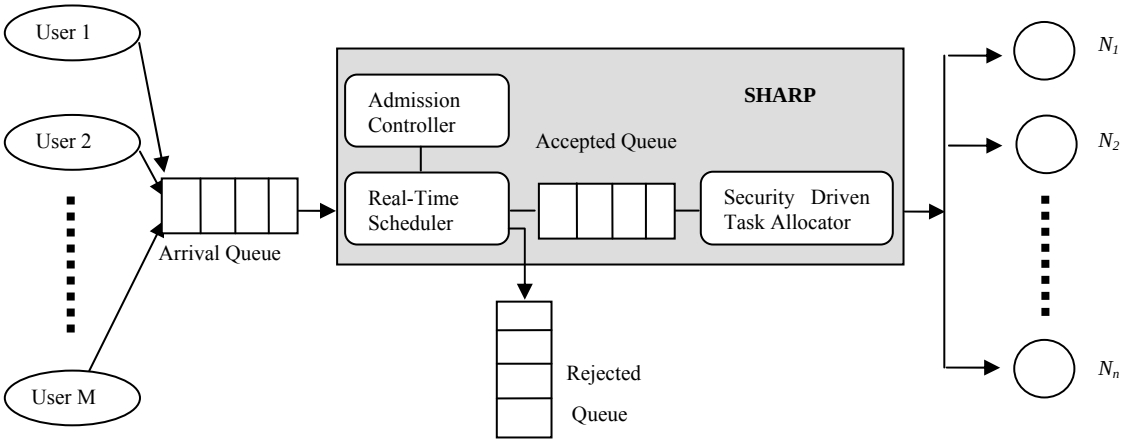


Figure 7.1 Scheduling architecture

7.2.2 Modeling Security Heterogeneity

We consider a class of embarrassing parallel applications (see [91] for some examples) each of which can be envisioned as a set of tasks without any interaction between one another. Our security driven scheduling approach can be easily extended to integrate a message scheduler, thereby being able to support communication-intensive parallel applications.

A parallel application is modeled as a tuple *(T, a, f, d, l)*, where $T = \{t_1, t_2, ..., t_m\}$ represents a set of tasks, *a* and *f* are the arrival and finish times, *d* is the specified deadline, and *l* denotes the amount of data (measured in MB) to be protected. Each task $t_i \in T$ is labeled with a pair, e.g., $t_i = (E_i, S_i)$, where E_i and S_i are vectors of execution times and security requirements for task t_i. The execution time vector denoted by $E_i = (e_i^1, e_i^2, \ldots, e_i^n)$ represents the execution time of t_i on each node in the cluster. Each task of a parallel application requires a set of security services providing various security levels,

which are normalized in the range from 0 to 1.0. Suppose $t_i \in T$ requires q security services, $S_i = (s_i^1, s_i^2, \ldots, s_i^q)$, a vector of security levels, characterizes the security requirements of the task. s_i^k $(1 \le k \le q)$ is the normalized security level of the kth security service required by t_i.

Let w_i^j denote the computational weight of task t_i on node N_j. w_i^j is computed as a ratio between its execution time on N_j and that on the fastest node in the cluster. Thus, we have $w_i^j = e_i^j \Big/ \max_{k=1}^{n}\left(e_i^k\right)$. The computational heterogeneity level of t_i, referred to as H_i^C, is quantitatively measured by the standard deviation of the computational weights. That is, H_i^C is expressed as:

$$H_i^C = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(w_i^{avg} - w_i^j\right)^2}\ , \text{where } w_i^{avg} = \left(\sum_{j=1}^{n} w_i^j\right) \Big/ n\,. \tag{7.1}$$

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

$$H^C = \frac{1}{n}\sum_{T_i \in T} H_i^C\ . \tag{7.2}$$

Besides computation heterogeneity, a cluster may exhibit security heterogeneity. While each task imposes requirements on a set of security services, each node in the cluster provides an array of security services with various quality of security, which is measured by security levels normalized in the range from 0 to 1.0. Security services provided by node N_j is characterized as a vector of security levels, $P_j = (p_j^1, p_j^2, \ldots, p_j^r)$, where p_j^k $(1 \le k \le r)$ is the security level of the kth security service provided by N_j.

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

$$H_i^S = \sqrt{\frac{1}{n}\sum_{j=1}^{q}\left(s_i^{avg} - s_i^j\right)^2}\ , \text{where } s_i^{avg} = \left(\sum_{j=1}^{q} s_i^j\right)\Big/ n . \tag{7.3}$$

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

$$H^S = \frac{1}{n}\sum_{T_i \in T} H_i^S\ . \tag{7.4}$$

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

$$H_k^V = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(p_{avg}^k - p_i^k\right)^2}\ , \text{where } p_{avg}^k = \left(\sum_{i=1}^{n} p_i^k\right)\Big/ n . \tag{7.5}$$

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

$$H_j^M = \sqrt{\frac{1}{n}\sum_{k=1}^{n}\left(p_j^{avg} - p_j^k\right)^2}\ , \text{where } p_j^{avg} = \left(\sum_{k=1}^{n} p_j^k\right)\Big/ n . \tag{7.6}$$

Using Equation (7.5), the heterogeneity in security services of the cluster can be computed as:

$$H^V = \frac{1}{n}\sum_{k=1}^{q} H_k^V\ . \tag{7.7}$$

7.2.3 Heterogeneity in Security overhead

Now we consider heterogeneity in security overhead. The security overhead model in [97] accounts for three security services, including encryption, integrity, and

authentication. The security overhead model can be easily extended to incorporate other security services.

Suppose task t_i requires q security services. Let s_i^k and $c_{ij}^k(s_i^k)$ be the security level and overhead of the kth security service, the security overhead c_{ij} experienced by t_i on node N_j can be computed using Equation (7.8) in general and Equation (7.9) in particular.

$$c_{ij} = \sum_{k=1}^{q} c_{ij}^k(s_i^k), \text{ where } s_i^k \in S_i, \tag{7.8}$$

$$c_{ij} = \sum_{k \in \{a, e, g\}} c_{ij}^k(s_i^k), \text{ where } s_i^k \in S_i, \tag{7.9}$$

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 [97]. The security overhead of an application with task set T is calculated by:

$$c = \sum_{t_i \in T} \sum_{j=1}^{n} x_{ij} c_{ij} = \sum_{t_i \in T} \sum_{j=1}^{n} \left(x_{ij} \sum_{k \in \{a, e, g\}} c_{ij}^k(s_i^k) \right), \tag{7.10}$$

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

7.2.4 Modeling Quality of Security

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

$$g(s_i^k, p_j^k) = \begin{cases} 0, \text{if } s_i^k \le p_j^k \\ s_i^k - p_j^k, \text{otherwise} \end{cases}. \tag{7.11}$$

A small SD value indicates a high degree of service satisfaction. In particular, a zero SD value implies that t_i's requirement placed on the kth security service can be perfectly met. The SD value of task t_i on node N_j can be derived from Expression (7.12). Thus, $SD(s_i, P_j)$ of t_i is computed as a weighted sum of the SD values of q required security services. Formally, we have:

$$SD(s_i, P_j) = \sum_{k=1}^{q} \left[w_i^k \, g\left(s_i^k, p_j^k\right) \right], \tag{7.12}$$

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

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

$$\begin{aligned} DSD(T, X) &= \sum_{t_i \in T} \sum_{j=1}^{n} \left[x_{ij} SD\left(s_i, P_j\right) \right] \\ &= \sum_{t_i \in T} \sum_{j=1}^{n} \left\{ x_{ij} \sum_{k=1}^{q} \left[w_i^k \, g\left(s_i^k, p_j^k\right) \right] \right\}, \end{aligned} \tag{7.13}$$

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

7.3 The SHARP Algorithm

Before we proceed to present the security driven scheduling algorithm, we formulate the scheduling problem as follows.

7.3.1 Security Driven Scheduling Problem Formulation

Let X be a schedule for task set T, we have $\forall x_{ij} \in X_i, \sum_{j=1}^{n} x_{ij} = 1 : x_{ij} = 1$ if $t_i \in T$ is allocated to node N_j. The scheduling decision of the task set T is optimal if (1) all the tasks can be completed before the deadline d, and (2) the degree of security deficiency (see Equation 7.13) is minimized. Since a security driven scheduler makes an effort to reduce the degree of security deficiency of each submitted parallel application, the following function needs to be minimized:

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

$$\text{subject to } f_i \le d, \ t_i \in T \tag{7.14}$$

where f_i is the finish time of the ith task in the task set.

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

$$\text{minimize} \sum_{\text{for all T}} DSD(T, X). \tag{7.15}$$

Substituting Equation (7.14) into (7.15) yields the following security objective function. Thus, our SHARP algorithm strives to schedule applications in a way to minimize the average degree of security deficiency of all schedulable parallel applications:

$$DSD = \sum_{\text{for all T}} \left\{ P_{sd}(T) \sum_{t_i \in T} \sum_{j=1}^{n} \left[x_{ij} \sum_{k=1}^{q} \left[w_i^k \, g\left(s_i^k, p_j^k\right) \right] \right] \right\} \Big/ \sum_{\text{for all T}} P_{sd}(T), \tag{7.16}$$

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

7.3.2 Property of the Algorithm

Now we list an important property of the SHARP algorithm, which will be used in the subsequent subsection to construct the security driven scheduling algorithm. The schedule of a parallel application is feasible if all tasks in its task set can be completed before its deadline. Thus, a parallel application has a feasible schedule on a heterogeneous cluster if there exists a set of nodes, where a valid schedule can be generated for each task. This fact can be express by the following property.

Property 7.1. If a parallel application with task set T and deadline d has a feasible schedule on a heterogeneous cluster with n sites denoted by a set N, the following inequality must be satisfied:

$$\forall t_i \in T : \exists N_j \in N : \sigma_i^j + e_i^j + \sum_{k=1}^{q} c_{ij}^k(\bar{s}_i^k) \le d \text{ , and } \bar{s}_i^k = \begin{cases} s_i^k, \text{if } s_i^k \le p_j^k \\ p_j^k, \text{otherwise} \end{cases},$$

where σ_i^j is the start time of the ith task on node N_j, e_i^j is the computation time of the ith task on N_j and $\sum_{k=1}^{q} c_{ij}^k(\bar{s}_i^k)$ is the security overhead experienced by the task. Note that

the security overhead of the *k*th security service relies on the security level $\bar{s}_i^k$, which equals to s_i^k if node N_j can ensure high quality of the *k*th security service to satisfy the task's security needs. $\bar{s}_i^k$ becomes p_i^k when N_j fails to fulfil t_i's security requirements. Property 7.1 formally describes timing constraints.

The earliest start time σ_i^j is one of the key factors in Property 7.1, and σ_i^j can be computed as:

$$\sigma_i^j = \tau + \sum_{t_l \in m.d_l \le d} \left(e_l^j + \sum_{k=1}^{q} c_l^k (\bar{s}_l^k) \right) \tag{7.17}$$

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

7.3.3 Algorithm Description

The SHARP algorithm is outlined in Figure 7.2. The goal of the algorithm is to deliver high quality of security under two conditions: (1) deadlines of submitted parallel applications are met; and (2) the degree of security deficiency (see Equation 7.16) of each admitted parallel application is minimized.

Before reducing the security deficiency value of each task of a parallel application, SHARP makes an effort to meet the timing constraint of the application. This can be accomplished by calculating the earliest start time (see Equation 7.17) and the security overhead of each task in Steps 5 and 6, followed by checking if all the tasks of the application can be completed before its deadline *d* (see Step 7). If there is no way of

guaranteeing the deadline of a task in the parallel application, the application is rejected by Step 16.

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

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

Figure 7.2 The SHARP scheduling algorithm

Now we derive the time complexity of the SHARP scheduling algorithm as follows.

Theorem 7.1. The time complexity of SHARP is *O(nmq)*, where *n* is the number of nodes in a cluster, *m* is the number of tasks in a parallel application, and *q* is the number of security services.

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

7.4 Evaluation of Security Risks

In this section, we first derive the probability $P_{rf}(t_i, N_j)$ that t_i remains risk-free during the course of its execution on node N_j. The risk-free probability of t_i with respect to the *k*th security service is:

$$P_{rf}^{k}(t_i, N_j) = \exp\left(-\lambda_i^k\left(e_i^j + \sum_{l=1}^{q} c_{ij}^{l}(s_i^l)\right)\right), \tag{7.18}$$

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. The risk rate is expressed as:

$$\lambda_i^k = 1 - \exp\left(-\alpha(1 - s_i^k)\right). \tag{7.19}$$

It is assumed 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, and the model can be replaced by any risk rate model with a reasonable parameter α (α is set to 0.002 in our experiments).

The risk-free probability of task T_i on node N_j can be written as Equation 7.20, where all security services provided to the task are considered. Thus, we have:

$$\begin{aligned} P_{rf}(t_i, N_j) &= \prod_{k=1}^{q} P_{rf}^{k}(t_i, N_j) \\ &= \prod_{k=1}^{q} \exp\left(-\lambda_i^k \left(e_i^j + \sum_{l=1}^{q} c_{ij}^l (s_i^l) \right) \right) \\ &= \exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l (s_i^l) \right) \sum_{k=1}^{q} \lambda_i^k \right). \end{aligned} \tag{7.20}$$

Using equation (7.20), we can write the overall risk-free probability of task t_i in the cluster as:

$$\begin{aligned} P_{rf}(t_i) &= \sum_{j=1}^{n} \left\{ P[x_{ij} = 1] \cdot P_{rf}(t_i, N_j) \right\}, \\ &= \sum_{j=1}^{n} \left\{ p_{ij} \cdot \exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l (s_i^l) \right) \sum_{k=1}^{q} \lambda_i^k \right) \right\} \end{aligned} \tag{7.21}$$

where p_{ij} is the probability that t_i is allocated to N_j. Given an application with task set T, the probability that all tasks are free from being attacked is computed as below:

$$\begin{aligned} P_{rf}(T) &= \prod_{T_i \in T} P_{rf}(T_i) \\ &= \prod_{T_i \in T} \left\{ \sum_{j=1}^{n} \left\{ p_{ij} \cdot \exp\left(-\left(e_i^j + \sum_{l=1}^{q} c_{ij}^l (s_i^l) \right) \sum_{k=1}^{q} \lambda_i^k \right) \right\} \right\}. \end{aligned} \tag{7.22}$$

Finally, we can calculate the average risk-free probability of all schedulable applications using Equation 7.23, where $P_{sd}(T)$ is a step function, and

$P_{sd}(T) = \begin{cases} 1, \text{if } T \text{ can be timely completed} \\ 0, \text{otherwise} \end{cases}$. It is worth noting that SHARP is conducive to maximizing the risk-free probabilities of heterogeneous clusters.

$$P_{rf} = \left(\sum_{\text{for all T}} P_{sd}(T) \cdot P_{rf}(T) \right) \Big/ \sum_{\text{for all T}} P_{sd}(T)$$

$$= \left\{ \sum_{\text{for all T}} \left[P_{sd}(T) \prod_{T_i \in T} P_{rf}(T_i) \right] \right\} \Big/ \sum_{\text{for all T}} P_{sd}(T) \qquad (7.23)$$

The risk-free probability computed by Equation 7.23 is used in concert with the degree of security deficiency (see Equation 7.16) to measure quality of security provided by a heterogeneous cluster. In the subsequent section, we quantitatively evaluate the risk-free probability and degree of security deficiency for heterogeneous cluster under a wide range of workload environments.

7.5 Performance Results and Comparison

Now we are in a position to evaluate the effectiveness of SHARP. To reveal the strength of SHARP, we compared it with two well-known real-time scheduling algorithms, namely, *EDF* (Earliest Deadline First) and *LLF* (Least Laxity First). While EDF schedules a ready job with the earliest deadline, LLF assigns priority based on jobs' laxities(laxity=Deadline–computation time). Table 7.1 summarizes key parameters of a simulated cluster used in our experiments.

7.5.1 Simulation Parameters

Before presenting the empirical results in detail, we present the simulation model as follows. The parameters of nodes in the simulated cluster are chosen to resemble real-world workstations like IBM SP2 nodes. We made use of a real world trace (e.g., San

Diego Supercomputer Center SP2 log sampled on a 128-node cluster) to conduct simulations. We modified the trace by adding a block of security-sensitive data for each task. "job number", "submit time", "execution time" and "number of requested processors" of jobs submitted to the system are taken directly from the trace. "deadlines", "security requirements of jobs", and "security-sensitive data size" are synthetically generated, since these parameters are not available in the trace. The performance metrics we used include: *Average risk-free probability* (see Equation 7.23), *Average degree of security deficiency* (see Equation 7.16), *Guarantee Ratio* (measured as a fraction of total submitted applications that are schedulable).

Table 7.1 Characteristics of System Parameters

Parameter	Value (Fixed) - (Varied)
Number of tasks	(6400) – The first three month trace data from SDSC
Number of nodes	(64) – (32, 64, 128, 192, 256)
CPU Speedup	(1) – (2, 3, 4, 5, 6, 7, 8, 9, 10)
Laxity	(4) – (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50) ×10000seconds
Job arrival rate	Decided by the trace
Data to be secured (uniform dist.)	(1–100) – (0.01–1,0.1–10,1–100,10–1000,100–10000,1000–100000) MB
Site security level (uniform dist.)	(0.1 – 1.0)
Application 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,
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)

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

7.5.2 Overall Performance Comparisons

Figure 7.3 shows simulation results for the three algorithms on a cluster with 64 nodes. We observe that SHARP significantly outperforms the two competitors. For example, SHARP increases the risk-free probability by an average of 164.25% (see Figure 7.3a), whereas SHARP achieves improvement in degree of security deficiency by an average of 357.2% (see Figure 7.3b). We attribute the security improvement of SHARP over EDF and LLF to the fact that SHARP judiciously assigns tasks of a parallel application to a group of nodes minimizing execution times while increasing security. With regard to guarantee ratio, SHARP is noticeably better than EDF and LLF for all tested cases. For example, the average improvement in guarantee rate is 15.8% (see Figure 7.3c). This is because for each arrived application SHARP selects a set of nodes that shorten total execution times.

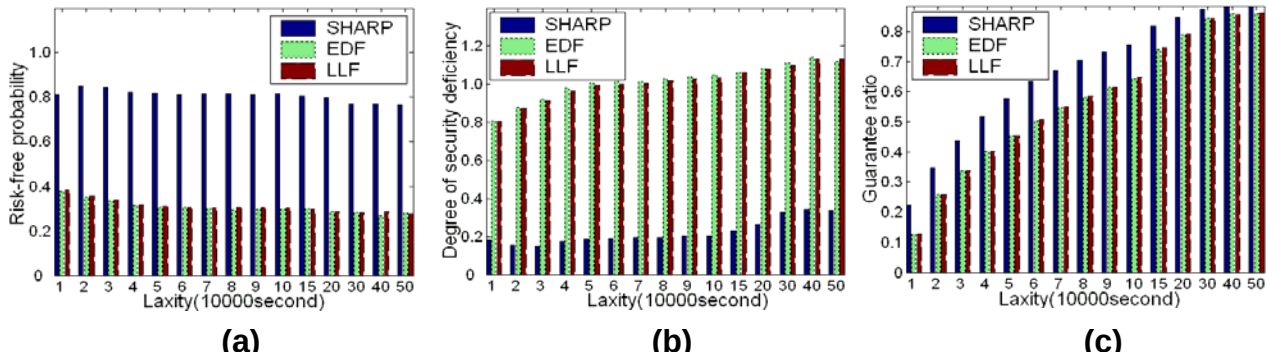


Figure 7.3 Performance impact of deadline

When laxity goes up, the values of degree of security deficiency increase (see Figure 7.3b). This is because SHARP can admit more applications when deadlines become loose. Under a high workload condition, however, there exist fewer candidate nodes that can fulfil security demands while timely completing accepted applications.

We notice in Figure 7.3a that the risk-free probability of SHARP slightly decreases with the increasing value of laxity. This result can be explained by two facts: (1) the risk-free probability is derived from both degree of security deficiency and total execution time (see Equation 7.23), and (2) a high degree of security deficiency results in a low value of risk-free probability

7.5.3 Impact of heterogeneities in security and computation

In this subsection we examine the performance impact of heterogeneities in security and computation on SHARP. The five heterogeneity configurations are detailed in Table 7.1. In recognition that deadline is a critical performance metric for a real-time job scheduler, we tested three deadline scenarios: loose, normal, and tight deadlines in this group of experiments.

In the case where the cluster is homogeneous (see the first configuration), SHARP noticeably outperforms EDF and LLF in security. However, SHARP delivers the same guarantee ratio performance as those of the two alternatives when the cluster is homogeneous.

Figure 7.4 shows that SHARP fully exhibits its strength when the security and computational heterogeneities increase. In particular, SHARP performs substantially better than the other two algorithms for the last four heterogeneity configurations. Additionally, it is observed that the risk-free probabilities and degrees of security deficiency of EDF and LLF only marginally change with the increasing values of security and computational heterogeneities. This is because EDF and LLF are unaware of security and, thus, merely assign tasks to nodes shortening computational times.

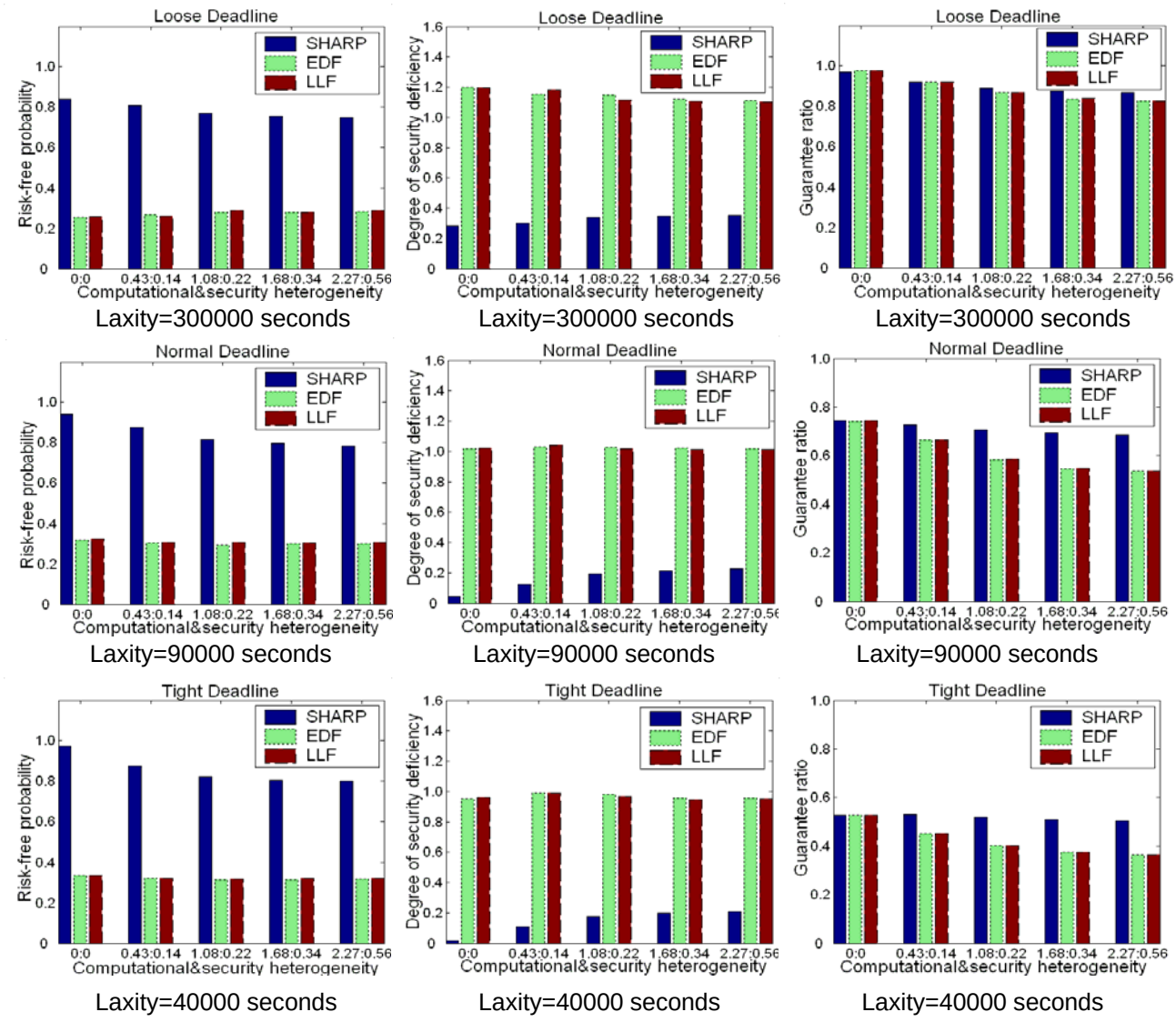


Figure 7.4 Performance impact of security and computational heterogeneities

It is intriguing to illustrate the impact of deadline in this set of experiments. When deadlines are loose, SHARP marginally improves guarantee ratio performance over EDF and LLF. However, when deadlines are normal or tight, SHARP is significantly superior to EDF and LLF in terms of guarantee ratio. The implication behind this result is that SHARP is the most appropriate algorithm for scenarios where parallel applications have tight deadlines.

7.5.4 Security-sensitive data size

In this set of experiments we evaluated the performance impact of security-sensitive data size. We tested six configurations of data size (see Table 7.1). The laxity is chosen to be 1000 seconds.

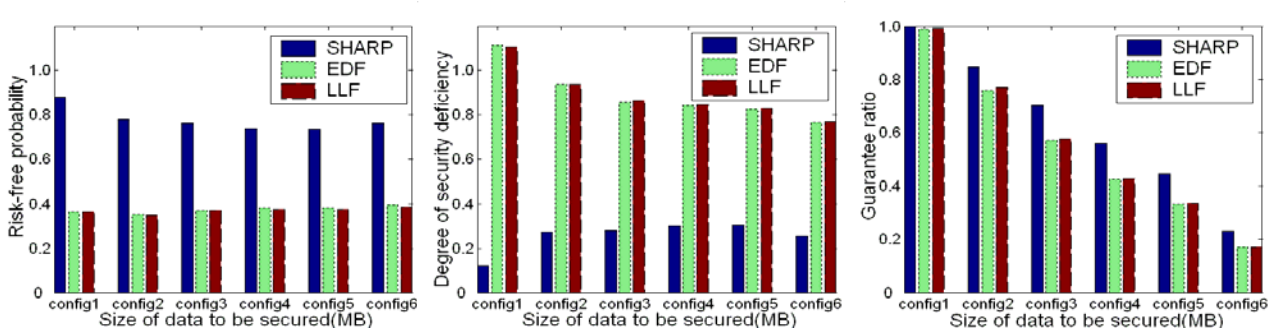


Figure 7.5 Performance impact of size of data to be secured

The experimental results are shown in Figure 7.5. There are several important observations. Firstly, when the security-sensitive data size increases, the degree of security deficiency of SHARP slightly increases. Unlike SHARP, the degrees of security deficiency of EDF and LLF marginally reduce with the increasing value of data size. Secondly, Figure 7.5 shows that the risk-free probability of SHARP slightly decreases with growing size of security-sensitive data. This is because (1) the degree of security deficiency increases, and (2) the total execution times become longer (see Equations 7.19 and 7.21). Conversely, the risk-free probabilities of EDF and LLF remain almost unchanged. Thirdly, when the data size is large, the performance improvement achieved by SHARP becomes more dramatic. This result suggests that SHARP can be employed to improve security of data-intensive applications.

7.5.5 Scalability

This group of experiments is intended to investigate the scalability of the SHARP algorithm. We scale the number of nodes in a heterogeneous cluster from 32 up to 256.

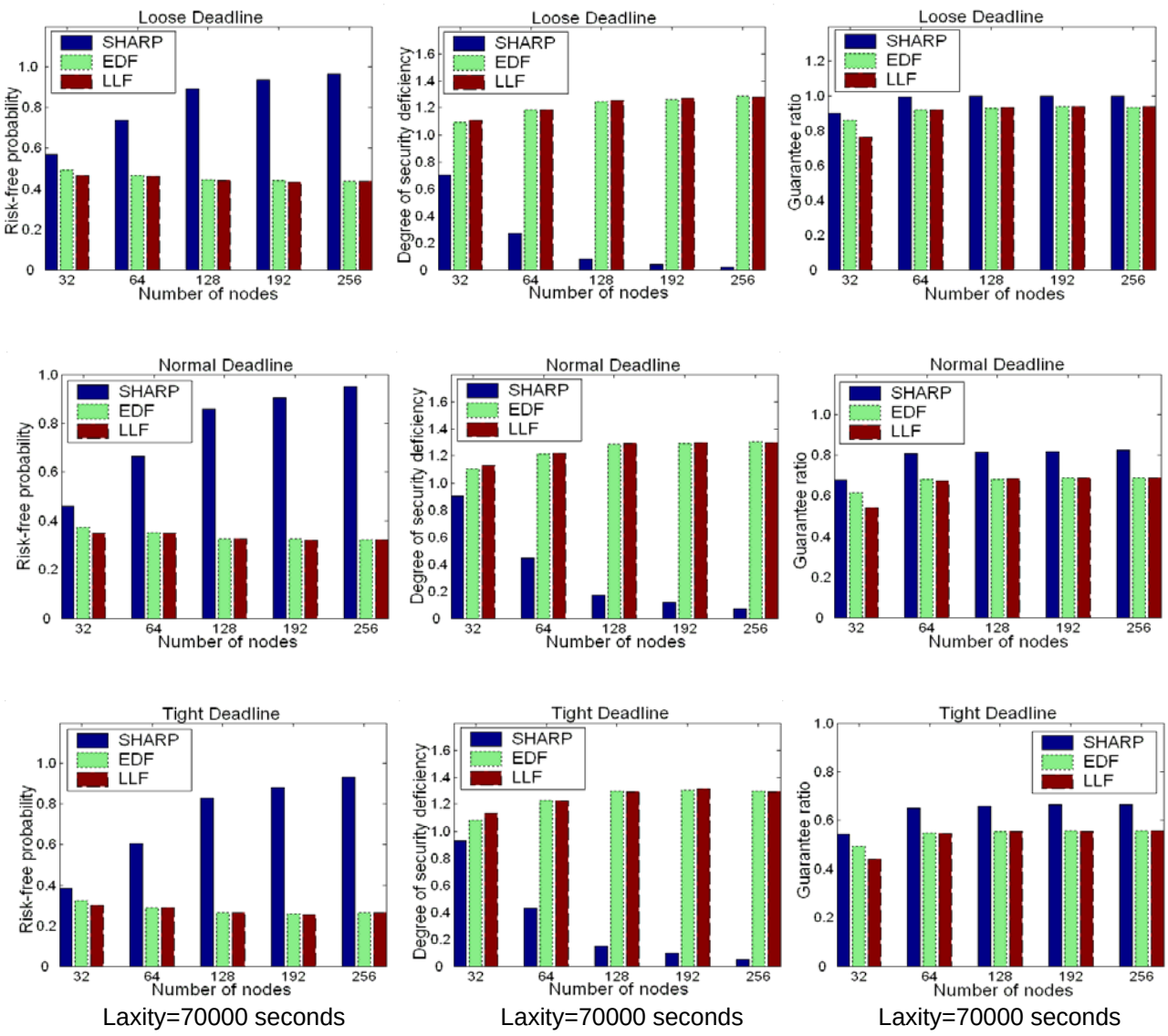


Figure 7.6 Performance impact of number of nodes

First, Figure 7.6 clearly shows that SHARP exhibits good scalability, and SHARP outperforms the other two alternatives in all three performance metrics for all cases. Second, it is observed from Figure 7.6 that SHARP makes more prominent improvement

in degree of security deficiency and risk-free probability when the heterogeneous cluster size scales up. Importantly, SHARP can achieve high performance for large-scale clusters. This is because there is a strong likelihood that SHARP can meet parallel applications' security demands while minimizing total execution times.

7.5.6 CPU Capacity

To examine security and performance sensitivities of the three algorithms to CPU capacities, in this set of experiments we varied the CPU capacity (measured as speedup over the baseline computational node) from 2 to 10. Specifically, the CPU speed of the IBM SP2 66MHz node is normalized to 1. We normalized the CPU capacity of the nodes to the values of 2, 3, 4, 5, 6, 7, 8, 9, and 10, respectively. The laxity is set to 10000 seconds, and the number of nodes is fixed to 32.

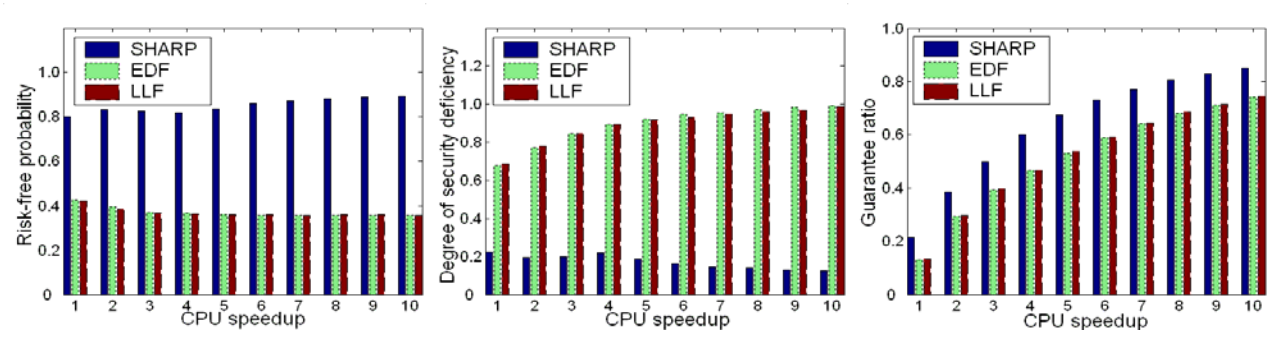


Figure 7.7 Performance impact of CPU speedup

The results presented in Figure 7.7 reveal that SHARP is superior to the other two competitors in all the three performance metrics. In addition, the improvements of SHARP in degree of security deficiency and risk-free probability become more prominent when the CPU capacity increases. The results imply that SHARP can gain additional security and performance benefits with increasing CPU capacities. These results as well as those presented in Section 7.5.5 clearly indicate that SHARP can be

used to improve both security and performance of parallel applications on large-scale heterogeneous clusters with powerful CPUs.

7.6 Summary

In this chapter, we presented a security- and heterogeneity-driven scheduling algorithm for real-time parallel applications running on heterogeneous clusters. The main contributions of this part of the dissertation study include: (1) an analysis of parallel applications with security and timing constraints; (2) two new performance metrics (degree of security deficiency and risk-free probability) used to quantitatively measure quality of security provided by heterogeneous clusters; (3) considerations of heterogeneities in security and computation; (4) an security-and heterogeneity-driven scheduling algorithm (SHARP for short); (5) a simulation tool for heterogeneous clusters where SHARP is implemented and evaluated.

Simulation results show that the degree of security deficiency can be used to increase risk-free probabilities of parallel applications. Our experimental results confirm that the SHARP algorithm can be employed to improve both security and performance of real-time parallel applications running on heterogeneous clusters. Furthermore, security and performance improvements of SHARP over two existing algorithms are more pronounced for large-scale heterogeneous clusters with powerful CPUs.

In the next Chapter, we will study the issue of security-aware real-time scheduling on computational grids.

Chapter 8

8 Enhancing Security of Real-Time Applications on Grids

In previous chapters, we concentrated on security-aware scheduling for real-time applications on clusters. As we mentioned earlier, computational Grids are emerging as next generation computing platforms for large-scale computation and data intensive problems in industry, academic, and government organizations. Therefore, in this chapter we aim at developing security-aware real-time scheduling strategies in the context of Grid computing.

In what follows, we summarize the motivation of this work in Section 8.1. Section 8.2 describes a preliminary system architecture and task model. In Section 8.3 we propose a real-time scheduling algorithm for parallel applications on Grids. We present in Section 8.4 the experimental results based on real world traces from a supercomputing centre. We also provide insight into system parameters that ultimately affect performance potential of SAREG. Finally, Section 8.5 concludes the chapter with summary.

8.1 Motivation

A computational grid is a collection of geographically dispersed computing resources, providing a large virtual computing system to users. With rapid advances in processing power, network bandwidth, and storage capacity, Grids are emerging as next generation computing platforms for large-scale computation and data intensive problems in industry, academic, and government organizations. As typical scientific simulation and computation require a large amount of compute power, it becomes crucial to take advantage of application scheduling to enable the sharing and aggregation of geographically distributed resources to meet the needs of highly complex scientific problems [63].

Recently there have been some efforts devoted to the development of real-time applications on Grids [29]. Real-time applications depend not only on results of computation, but also on time instants at which these results become available [35]. The consequences of missing deadlines of hard real-time systems may be catastrophic, whereas such consequences for soft real-time systems are relatively less damaging. Examples of hard real-time applications include distributed defense and surveillance applications [82], and distributed medical data systems [49]. On-line transaction processing systems are examples of soft real-time applications [57].

There exist a growing number of systems that have real time and security considerations [75], because sensitive data and processing require special safeguard and protection against unauthorized access. In particular, a variety of motivating real-time applications running on Grids require security protections to completely fulfill their security-critical needs. Unfortunately, conventional wisdom on real time systems is

inadequate for applications with real-time and security requirements. This is mainly because traditional real-time systems are developed to guarantee timing constraints while possibly posing unacceptable security risks.

To tackle the aforementioned problem, we proposed a real-time scheduling algorithm (referred to as SAREG) with security awareness, which is intended to seamlessly integrate security into real-time scheduling for applications running on Grids. In this chapter we use trace-driven simulation to compare the performance of the SAREG algorithm against three baseline scheduling algorithms for Grids. Our simulator combines performance and security overhead estimates using a security overhead model based on three most commonly used security services (see Chapter 4).

8.2 Mathematical Model

A mathematical model of security-aware real-time scheduling for Grids is presented in this section. This model, which describes a scheduling framework, security-sensitive real-time jobs, and security overheads, allows the SAREG algorithm to be formally presented in Section 8.3.

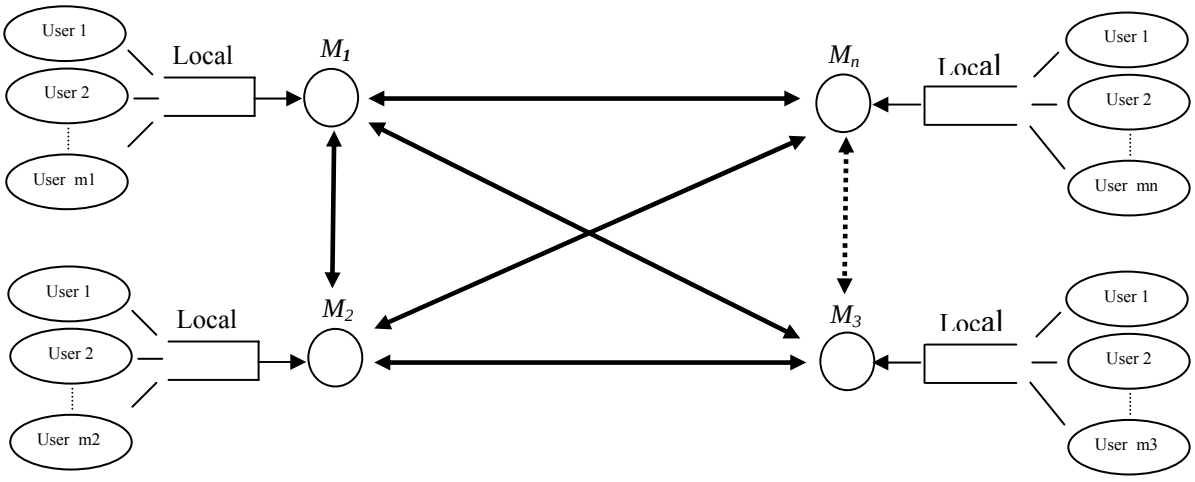


Figure 8.1 Scheduling framework for SAREG in a computational Grid

8.2.1 Scheduling Framework

A Grid can be specified as $G = \{M_1, M_2, \ldots, M_n\}$, where M_i, $1 \leq i \leq n$, is a site or cluster [66]. The n sites are connected by wide-area networks (See Figure 8.1). The scheduling framework is general in the sense that it can be applied to small-scale grids where computational sites are connected by LAN or MAN. Each site M_i is represented as a vector, e.g., $M_i = (P_i, N_i, T_i, Q_i)$, where P_i is the peek computational power measured by an overall CPU capacity (e.g., BIPS), N_i is the number of machines in the site, T_i is a set of accepted jobs running on M_i, and Q_i is a scheduler. Note that there exists a scheduler in each site, and we advocate the use of a distributed scheduling framework rather than a centralized one. This is mainly because a centralized scheduler in a large-scale grid inevitably becomes a severe performance bottleneck, resulting to a significant performance drop when workload is high. The distributed scheduling infrastructure makes a system portable, secure, and capable of distributing scheduling workload among an array of computational sites in the system [62].

Each site continues to receive reasonably up-to-date global load information by monitoring resource utilization of the Grid, and periodically broadcasts it local load information to other sites of the Grid. When a real-time job is submitted by a user to a local site, the corresponding scheduler assigns the job to a group of local machines or migrate the job to a remote site within in the Grid. The scheduler consists of a s*chedule queue* used to accommodate incoming real-time jobs. The scheduler queue is maintained by an admission controller. If the incoming real-time jobs can be scheduled, the admission controller will place the tasks in the *accepted queue* for further processing. In case no site can guarantee the deadline of the submitted real-time job, it will be dropped

into a local *rejected list*. The scheduler processes all the accepted tasks by its scheduling policy before transmits them into the *dispatch queue*, where the quality of security of accepted jobs are maximized. After the quality of security is enhanced, the real-time job is dispatched to one of the designated site $M_i \in G$. The machines in site M_i can execute a number of real-time tasks in parallel.

8.2.2 Security-Sensitive Real-time Jobs

A real-time job is specified as a set of rational parameters, e.g., $J_i = (e_i, p_i, d_i, l_i, S_i)$, where e_i is the execute time, p_i is the number of machines required to execute J_i, d_i is the deadline, and l_i denotes the amount of data (measured in KB) to be protected. e_i can be estimated by code profiling and statistical prediction [16]. A collection of security services required by J_i is specified as $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, where S_i^j denotes the security level range of the jth security service. Our security-aware scheduler is intended to determine the most appropriate point s_i in space S_i, e.g., $s_i = (s_i^1, s_i^2, \ldots, s_i^q)$, where $s_i^j \in S_i^j$, $1 \le j \le q$.

A schedule of a job J_i is formally denoted as the following expression:

$$x_i = \left((\sigma_{i,1}, s_{i,1}), (\sigma_{i,2}, s_{i,2}), \ldots (\sigma_{i,pi}, s_{i,pi})\right), \tag{8.1}$$

where J_i is divided into p_i tasks, $\sigma_{i,j}$ and $s_{i,j}$ are the start time and the security level of the jth task.

The SAREG algorithm is able to measure the security benefits gained by each admitted job. To implement this basic and valuable functionality, we quantitatively model the security benefit of the jth task of job J_i as a security level function denoted by $SL: S_i \rightarrow \Re$, where $\Re$ is the set of positive real numbers:

$$SL(s_{i,j}) = \sum_{k=1}^{q} w_i^k s_{i,j}^k \text{ , where } s_{i,j} = \left(s_{i,j}^1, s_{i,j}^2, ..., s_{i,j}^q\right), 0 \le w_i^j \le 1 \text{, and } \sum_{j=1}^{q} w_i^j = 1. \quad (8.2)$$

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

$$SL(s_i) = \sum_{j=1}^{p_i} SL(s_{i,j}) \text{, where } s_i = (s_{i,1,} s_{i,2,}, ..., s_{i,pi}). \quad (8.3)$$

The scheduling decision of the job J_i is feasible if (1) all its tasks can be completed before the deadline d_i, and (2) the corresponding security requirements are satisfied. Specifically, given a real-time job J_i that consists of p_i tasks, we can obtain the following non-linear optimization problem formulation to compute the optimal security benefit of J_i:

$$\text{maximize } SL(s_i) = \sum_{j=1}^{p_i} \sum_{k=1}^{q} w_i^k s_{i,j}^k \text{ ,}$$

$$\text{subject to } \min(S_i^k) \le s_{i,j}^k \le \max(S_i^k),$$

$$f_{i,j} \le d_i \quad (8.4)$$

where $f_{i,j}$ is the finish time of the *j*th task of J_i, and $\min(S_i^j)$ and $\max(S_i^j)$ are the minimum and maximum security requirements of J_i.

The SAREG scheduling algorithm to be presented in the next section strives to enhance quality of security, which is defined by the sum of the security levels of all the

admitted jobs. Thus, the following security value function needs to be optimized, subject to certain timing and security constraints:

$$\text{maximize } SV = \sum_{j=1}^{n} \sum_{J_i \in T_j} y_{i,j} SL(s_i), \tag{8.5}$$

where T_j is a set of accepted jobs running on site M_j, $y_{i,j}$ is set to 1 if job J_i is accepted by the jth site, and is set to 0 otherwise. Substituting Equation (8.4) into (8.5) yields the following security value objective function. Thus, our job scheduling problem for Grid environments can be formally defined as follows: given a Grid $G = \{M_1, M_2, \ldots, M_n\}$ and a list of jobs submitted to the Grid, find a schedule $x_i = \left((\sigma_{i,1}, s_{i,1}), (\sigma_{i,2}, s_{i,2}), \ldots (\sigma_{i,pi}, s_{i,pi})\right)$ for each job J_i, such that the total security level of jobs on G,

$$SV = \sum_{i=1}^{n} \sum_{J_j \in T_i} y_{i,j} \sum_{k=1}^{p_j} \sum_{l=1}^{q} w_j^k s_{j,k}^l, \tag{8.6}$$

is maximized.

8.3 The SAREG Scheduling Algorithm

To support the presentation of the proposed algorithm, it is necessary to introduce some definitions as well as three properties. The schedule of a job is feasible if all its tasks can be completed before its deadline. Hence, a job has a feasible schedule on a Grid if there exists at least one site, where a valid schedule is available for each task. More formally, this fact can be expressed by the following property.

Property 8.1. If job J_i has a feasible schedule on a Grid with n sites denoted by a set $G = \{M_1, M_2, \ldots, M_n\}$, the following inequality must be satisfied:

$$\exists M_j \in G : \forall 1 \le k \le p_i : \sigma_{i,k}^{j} + e_{i,k} + c_i^{min} \le d_i,$$

where $\sigma_{i,k}^{j}$ is the start time of the *k*th task on site M_j, $e_{i,k}$ is the computation time of the *k*th task of J_i and c_i^{min} is the security overhead experienced by the task when its minimal security requirements are met. Property 8.1 formally describes the timing constraint, e.g., all the tasks of job J_i must be completed before the deadline d_i.

The start time can be computed by Equation (8.7).

$$\begin{cases} \sigma_{i,k}^{j} = \tau + \min\limits_{1 \le l \le N_i} \left\{ \sum\limits_{\substack{t_q \in m_l \\ d_q \le d_i}} \left(e_q + \sum\limits_{b \in \{a,e,g\}} c_q^b(s_q^b) \right) \right\}, \text{if task } t_k \text{is running on the local site } M_j \\ \sigma_{i,k}^{j,r} = \tau + \max\left\{ \dfrac{\delta_i^s}{B_{j,r}}, \min\limits_{1 \le l \le N_r} \left\{ \sum\limits_{\substack{t_q \in m_l \\ d_q \le d_i}} \left(e_q + \sum\limits_{b \in \{a,e,g\}} c_q^b(s_q^b) \right) \right\} \right\}, \text{if } t_k \text{ is dispatched to a remote site } M_r \end{cases} \quad (8.7)$$

where τ is the current time, δ_i^s is the size of data to be migrated from site M_j to M_r, $B_{j,r}$ denotes the available network bandwidth between M_j and M_r, and $\sum\limits_{t_q \in m_l, d_q \le d_i} \left(e_q + \sum\limits_{b \in \{a,e,g\}} c_q^b(s_q^b) \right)$ is the overall execution time (security overhead is factored in) of all tasks running on the *l*th machine. In other words, if task t_k is running on the local site M_j, the start time $\sigma_{i,k}^{j}$ is the earliest available time at site M_j. In case is t_k dispatched to be executed on the remote site M_r, the start time $\sigma_{i,k}^{j,r}$ relies on the data communication time and the earliest available time at M_r.

The minimal security overhead $c_{i,k}^{min}$ of t_i can be efficiently calculated by the

following equation.

$$c_i^{min} = \sum_{j \in \{a,e,g\}} c_i^j \left(\min \left\{ S_i^j \right\} \right), \tag{8.8}$$

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

Given an arrival job J_i and a local site M_j ($M_j \in G$) of the Grid, the scheduling problem is to generate a feasible schedule, which satisfies the following two properties.

Property 8.2. All the tasks of job J_i are accomplished before the deadline d_i. Thus,

$$\forall 1 \le k \le p_i : \begin{cases} \sigma_{i,k}^j + e_{i,k} + \sum_{b \in \{a,e,g\}} c_{i,k}^b (s_{i,k}^b) \le d_i, \text{ if task } t_k \text{ is running on the local site } M_j \\ \sigma_{i,k}^j + e_{i,k} + \sum_{b \in \{a,e,g\}} c_{i,k}^b (s_{i,k}^b) + \frac{\delta_i^r}{B_{r,j}} \le d_i, \text{ if } t_k \text{ is dispatched to a remote site } M_r \end{cases}$$

where $s_{i,k}^j \in S_i^j$ is the security level of the *j*th security service, δ_i^r is the size of output data to be transmitted from site M_r to M_j, and $B_{r,j}$ is the available network bandwidth between M_r and M_j.

Property 8.3. The security level of an accepted job J_i on M_j is maximized at the job's arrival time under the assumption that no more jobs arrive on M_j after this arrival time.

The *SAREG* algorithm is outlined in Figure 8.2. The goal of the algorithm is to deliver high quality of security under two conditions: (1) the security level promotion will not miss its deadline; and (2) the security level promotion will not result in any accepted subsequent task to be failed. To achieve the goal, *SAREG* strives to maximize security level of each accepted job (see Step 21) while maintaining reasonably high guarantee ratios (see Step 12).

1. **for** each task t_k of job J_i on site M_j **do**
2. Use **Equation (11)** to computer $\sigma_{i,k}^{j}$, the earliest start time of task J_i on site M_j;
3. Use **Equation (12)** to obtain the minimal security overhead $c_{i,k}^{min}$ of task t_k;
4. **if** $\sigma_{i,k}^{j} + e_{i,k} + c_{i,k}^{min} \le d_i$ **then** (See **Property 1**)
5. Sort the security service weights in a decreasing order of their values, e.g., $w_i^{v_1} < w_i^{v_2} < w_i^{v_3}$, where $v_l \in \{a, e, g\}, 1 \le l \le 3$;
6. **for** each security service $v_l \in \{a, e, g\}, 1 \le l \le 3,$ **do**
7. $s_{i,k}^{v_l} = \min\{S_i^{v_l}\}$; /* Initialize the security value of security service v_l */
8. **for** each security service $v_l \in \{a, e, g\}, 1 \le l \le 3,$ **do**
9. **while** $s_{i,k}^{v_l} < \max\{S_i^{v_l}\}$ **do**
10. increase security level $s_{i,k}^{v_l}$;
11. Use **Equation (10)** to calculate the security overhead of t_k on M_j;
12. **if** $\sigma_{i,k}^{j} + e_{i,k} + \sum_{b \in \{a,e,g\}} c_{i,k}^{b}(s_{i,k}^{b}) > d_i$ (based on **Property 1**) **then**
13. decrease security level $s_{i,k}^{v_l}$; break;
14. **end while**
15. **end for**
16. $SL(s_{i,k}) = \sum_{b \in \{a,e,g\}} w_i^b s_{i,k}^b$ /* Obtain the security level of t_k on M_j using **Equation (2)** */
17. **else** Migrate t_k to another site M_r, subjecting to

$$\min_{1 \le r \le n, r \ne j} \left\{ \sigma_{i,k}^{j,r} + e_{i,k} + \sum_{b \in \{a,e,g\}} c_{i,k}^{b}(s_{i,k}^{b}) + \frac{\delta_i^r}{B_{r,j}} \right\} \le d_i$$

18. **end for**
19. **if Property 2** is satisfied **then /*** All the tasks in J_i can be finished before d_i */
20. $y_{i,h} \leftarrow 1$, where $h = j$ or r; /* Accept job J_i */
21. /* Optimize quality of security, see **Equation (4)** */
Find site M_h for J_i, maximize $SL(s_i) = \sum_{k=1}^{p_i} \sum_{b \in \{a,e,g\}} w_i^b s_{i,k}^b$;
22. dispatch job J_i to M_h according to the schedule generated above;
23. **else** $y_{i,h} \leftarrow 0$; /* Reject J_i, since no feasible schedule is available */

Figure 8.2 The SAREG scheduling algorithm

Before optimizing the security level of each task of job J_i on M_j, *SAREG* attempts to meet the real-time requirement of J_i. This can be accomplished by calculating the earliest

start time (use Equation 8.7) and the minimal security overhead of J_i (use Equation 8.8) in Steps 2 and 3, followed by checking if all the tasks of J_i can be completed before the deadline d_i (see Step 4). If the deadline cannot be met by M_j, J_i is rejected by Step 23.

The security level of each task in J_i on M_j is optimized in the following way. Recall that the security service weights used in Equations (8.2), (8.4), and (8.6) reflect the importance of the three security services, directly indicating that it is desirable to give higher priorities to security services with higher weights (see Step 5). In other words, enhancing security levels of more important services tends to yield a maximized security level of the task on M_j.

In the case of a particular security service $v_l \in \{a, e, g\}$, Step 10 escalates the security level $s_{i,k}^{v_l}$ while satisfying the following timing constraints (see Step 12). Step 21 is able to maximize the security level of all the tasks in J_i by identifying a site M_h that provides the maximal security level and dispatching J_i to M_h (see Step 22).

The time complexity of the *SAREG* scheduling algorithm is given as follows.

Theorem 8.1. The time complexity of *SAREG* is *O(knm)*, where *n* is the number of sites in a Grid, *m* is the number of tasks running on a site, 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 finding the earliest start time for the task on a site is *O(m)* (Step 2). To obtain the minimal security overhead c_i^{min} of the task; the time complexity is a constant *O(1)* (Step 3). Sorting the security service weights in a decreasing order (Step 5) takes a constant time *O(1)* since we only have 3 security services. To increase the task's three security levels to their possible maximal ranks under the constraints (Step

12), the worst case time complexity is *O(3km)* (Step 8 ~ Step 15). To find site M_h on which the security level of task is optimized (Step 20 ~ Step 22), the time complexity is *O(n)*. Thus, the time complexity of the *SAREG* algorithm is as follows: O(*n*)(*O(m)* + *O(1)* + *O(1)* + *O(3km))* + *O(n)* = *O(knm)*. □

Since *n*, *m* and *k* cannot be very big numbers in practice, the time complexity of *SAREG* should be low based on the expression above. This time complexity indicates that the execution time of *SAREG* is a small value compared with task execution times (e.g., the real world trace used in our simulations shows that the average job execution time is 8031 Sec.). Thus, the CPU overhead of executing *SAREG-EDF* is ignored in our experiments.

We prove the correctness of the *SAREG* algorithm below.

Theorem 8.2. The *SAREG* algorithm satisfies Properties 8.2 and 8.3.

Proof. (1) First, we prove that *SAREG* satisfies Property 2. A task t_k is accepted by a Grid with *n* sites denoted by $G = \{M_1, M_2, \ldots, M_n\} \Rightarrow$ There is at least one site M_j ($M_j \in G$) on which t_k has a feasible schedule $\overset{\text{Property1}}{\Rightarrow}$ The two inequalities in Property1 must hold $\overset{\text{inequality1}}{\Rightarrow}$ task t_k can be finished before its deadline $d_i \Rightarrow$ The deadline of task t_k must be met. Thus, each accepted task meets its deadline.

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

(a) Task t_k is the last element in the local queue of site M_j based on the EDF order. In this case, there is no other task in the local queue of site M_j that is behind t_k. The only constraint for increasing the security level of t_k is its deadline d_i, which is enforced by

Step 12 in Figure 8.2. The security level of task t_k will eventually reach a critical value SL_i^{jc1} (Step 8 ~ Step 15 in Figure 3), meaning that any further increase in security level of t_k will violate its deadline d_i. Now suppose that there is a larger security level SL_i^{jb1} ($SL_i^{jb1} > SL_i^{jc1}$) for task t_k that is an accepted task on site M_j. However, this SL_i^{jb1} will definitely make t_k violate its deadline d_i based on the conclusion drawn above because of the equality $SL_i^{jb1} > SL_i^{jc1}$. SL_i^{jb1} makes T_i violate its deadline $d_i \Rightarrow \sigma_{i,k}^{j} + e_{i,k} + \sum_{b\in\{a,e,g\}} c_{i,k}^{b}(s_{i,k}^{b}) > d_i \Rightarrow$ t_k cannot be accepted by site $M_j \Rightarrow$ This statement contradicts our assumption that task t_k is an accepted task on M_j. Thus, SL_i^{jc1} must be the maximal security level of t_i under this situation.

(b) Task t_k is not the last element in the local queue of site M_j based on the EDF order. In this case, there is at least one accepted task in the local queue that is behind t_k. The deadline d_i constraint is enforced by Step 12. The security level of task t_k will also eventually reach a critical value SL_i^{jc2} (Step 8 ~ Step 15 in Figure 8.2), which means that further increase in the security level of t_k will either violate its deadline d_i or the deadlines of earlier accepted tasks. Now suppose SL_i^{jc2} is not t_k's maximal security level under this circumstance and, thus, there is a larger security level SL_i^{jb2} ($SL_i^{jb2} > SL_i^{jc2}$) for task t_k, an accepted task on node M_j under this situation. However, SL_i^{jb2} will violate either deadline d_i or the deadlines of earlier accepted tasks because of the inequality $SL_i^{jb2} > SL_i^{jc2}$.

Case one: SL_i^{jb2} violates t_k's deadline $d_i \Rightarrow \sigma_{i,k}^{j} + e_{i,k} + \sum_{b\in\{a,e,g\}} c_{i,k}^{b}(s_{i,k}^{b}) > d_i \Rightarrow$ t_k cannot be accepted on site M_j, which contradicts our assumption that task t_k is an accepted task

on site M_j. Thus, SL_i^{jc2} must be the maximal security level of t_k under this situation.

Case two: SL_i^{jb2} violates the deadlines of earlier accepted tasks. Thus, $\exists t_k : \sigma_{i,k}^j + e_{i,k} + \sum_{b\in\{a,e,g\}} c_{i,k}^b(s_{i,k}^b) > d_i$. The implication is that the second inequality in Property 1 cannot hold. Therefore, task T_i has no feasible schedule on site M_j, meaning that t_k is not an accepted task on site M_j. This statement contradicts our assumption that t_k is an accepted task on site M_j. Consequently, SL_i^{jc2} must be the maximal security level of t_k under this situation. □

8.4 Performance Evaluation

In the previous Section we proposed the SAREG scheduling algorithm, which integrates security requirements into scheduling for real-time applications on Grids. Now we are in a position to evaluate the effectiveness of SAREG by conducting extensive simulations based on a real world trace from San Diego Supercomputer Center (SDSC SP2 log). 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 jobs in simulation.

To reveal the strength of SAREG, we compared it against two well-know scheduling algorithms, namely, *Min-Min* and *Sufferage* [52] in addition to a traditional real-time scheduling algorithm - the Earliest Deadline First algorithm (*EDF)*. Min-Min and Sufferage are non-preemptive task scheduling algorithms, which were designed to schedule a stream of independent tasks onto a heterogeneous distributed computing system such as a Grid. Note that Min-Min and Sufferage are representative dynamic scheduling algorithms for Grid environments, and they were successfully applied in real

world distributed resources management systems such as SmartNet [30]. For the sake of simplicity, throughout this section *Sufferage* is referred to as *SUFFER*.

To emphasize the non-security-aware characteristic of *EDF* algorithm, we refer to the EDF algorithm as *NS-EDF* (non-security-aware *EDF*) in this chapter. Although the *NS-EDF* algorithm, a variation of *EDF*, is able to schedule real-time jobs with security requirements, it makes no effort to optimize quality of security. Rather, it randomly selects a security level for each task in a real-time job. The three baseline scheduling algorithms are briefly described below.

(1) *MINMIN*: For each submitted real-time job, a Grid site that offers the earliest completion time is tagged. Among all the mapped tasks, the one that has the minimum earliest completion time is chosen and then allocate to the tagged site. The MINMIN scheduler randomly selects security levels of security services required by tasks of a real-time job.

(2) *SUFFER*: Allocating a site to a submitted job that would "suffer" most in terms of completion time if that site is not allocated to it. Again, the SUFFER scheduler randomly chooses security levels for security requirements posed by an arriving job.

(3) *NS-EDF*: Tasks with the earliest deadlines are always executed first. We modified the traditional *EDF* algorithm in a way that it can randomly picks values within the security level ranges of services required by tasks. The following expression is held in the NS-EDF algorithm: $\forall 1 \le k \le p_i, b \in \{a, e, g\} : s_{i,k}^b = \text{random}\{S_i^b\}$.

The ultimate goal of comparing SAREG against MINMIN and SUFFER is to demonstrate schedulability performance improvements over existing scheduling algorithms in a real-time computing environment, whereas the purpose of comparing

SAREG with *NS-EDF* is to show security performance benefits gained by employing SAREG in a Grid environment. This section is organized as follows. Section 8.4.1 describes our simulator and important system parameters. Section 8.4.2 is to examine the performance improvements of SAREG over the three baseline algorithms. In Section 8.4.3 we investigate the performance impacts of the number of computing nodes in a simulated four-site Grid. Section 8.4.4 addresses the performance sensitivity of the SAREG algorithm to CPU capacities of the nodes in a Grid. We evaluate in Section 8.4.5 the scalability (measured as Grid size) of the proposed SAREG algorithm. Last but not least, Section 8.4.6 demonstrates that SAREG delivers good performance in terms of conventional performance metrics, including the mean slowdown and mean response time.

8.4.1 Simulator and Simulation Parameters

Before presenting the empirical results in detail, we present the simulation model as follows. A competitive advantage of conducting simulation experiments is that performance evaluation on a Grid can be accomplished without additional hardware cost. The Grid simulator was designed and implemented based on the model and the algorithm described in the previous sections.

Table 8.1 summarizes the key configuration parameters of the simulated Grid used in our experiments. The parameters of nodes in Grid are chosen to resemble real-world workstations like IBM SP2 nodes.

We modified a real world trace by adding randomly generated deadlines for all tasks in the trace. The assignment of deadlines is controlled by a deadline base, or laxity,

denoted as β, which sets an upper bound on tasks' slack times. We use Equation (8.9) to generate job J_i's deadline d_i.

$$d_i = a_i + e_i + c_i^{max} + \beta, \tag{8.9}$$

where a_i and e_i are the arrival and execution times obtained from the real-world trace. c_i^{max} is the maximal security overhead (measured in ms), which is computed by Equation (8.10).

$$c_i^{\max} = \sum_{j \in \{a,e,g\}} c_i^j \left(\max\left\{S_i^j\right\}\right), \tag{8.10}$$

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

Table 8.1 Characteristics of System Parameters

Parameter	Value (Fixed) - (Varied)
CPU Speed	(2) – (4, 8, 16)
β (Deadline Base, or Laxity)	(50 second) – (200, 400, 800 second)
Network bandwidth	5 MB/Second
Number of sites	(4) – (8, 16, 32)
Number of nodes	(184)- (256, 320,384)
Mean size of data to be secured	50KB for short jobs, 500KB for medium jobs, 1MB for long jobs
Mean size of input data	100MB for short jobs, 500MB for medium jobs, 1TB for large jobs
Mean size of application code	500KB for short jobs, 5MB for medium jobs, 50MB for large jobs
Required security services	Encryption, Integrity and Authentication
Weight of authentication	0.2
Weight of encryption	0.5
Weight of integrity	0.3

"Job number", "submit time", "execution time" and "number of requested processors" of jobs submitted to the Grid are taken directly from the trace. "size of input file", "size of application code", "size of output file" and "deadlines" are synthetically

generated in accordance with the above model, since these parameters are not available in the trace. When a job has to be remotely executed to meet its deadline, we must consider its migration cost, which is factored in Equation 8.7. In order to measure the migration cost of a job, we need to estimate the network bandwidth and the amount of data to be transferred. Vazhkudai *et al.* [86] measured the end-to-end bandwidth between two remote super-computing centres using GridFTP. It was discovered that the network bandwidth varies from 1.5 to 10.2 MB/sec. In our simulation experiments, the network bandwidth was randomly drawn from a uniform distribution with range 1.5 to 10.2 MB/sec., which can resemble practical network bandwidth in existing distributed systems. The synthesized deadline weakens correlations between real-time requirement and other workload characteristics. However, in the experiments we can examine performance impacts of deadlines on system performance by controlling the deadlines as fundamental simulation parameters (see Section 8.4.2).

The performance metrics by which we evaluate system performance include:

- *security value* (see Equation 8.6).
- *guarantee ratio:* measured as a fraction of total submitted jobs that are found to be schedulable).
- *overall system performance:* defined as a product of security value and guarantee ratio.
- *mean slowdown:* the slowdown of a job is the ratio of a job's response time to its service time, and mean slowdown is the average slowdown of all schedulable jobs in the Grid.

- *mean response time:* the response time of a job is the time interval between the job's arrival and finish times, and mean response time is the average response time of all schedulable jobs in the Grid.

8.4.2 Overall Performance Comparisons

The goal of this experiment is two fold: (1) to compare the proposed SAREG algorithm against the three baseline schemes, and (2) to understand the sensitivity of SAREG to parameter β, or deadline base (Laxity). To stress the evaluation, we assume that each job arrived in the Grid requires the three security services. Without loss of generality, it is assumed that time spent handling page faults is factored in a job's execution time.

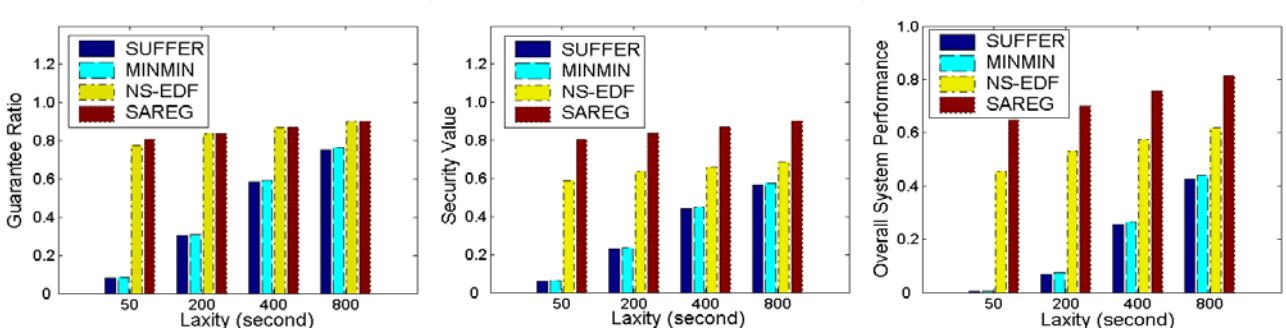


Figure 8.3 Performance impact of deadline

Figure 8.3 shows the simulation results for these four algorithms on a Grid with 4 sites (184 nodes) where the CPU power is fixed at 100 MIPS. We observe from Figure 8.3a that SAREG and NS-EDF exhibit similar performance in terms of guarantee ratio (the performance difference is less than 2%), whereas the guarantee ratios of SAREG are a lot higher than those of MINMIN and SUFFER algorithms. The reason for the performance improvements of SAREG over MINMIN and SUFFER is two fold. First, SAREG is a real-time scheduler, while the SHMAX and SHRND are non-real-time

scheduling algorithms. Second, SAREG judiciously enhances the security levels of accepted jobs under the condition that the deadlines of the accepted jobs are guaranteed.

Figure 8.3a illustrates that the guarantee ratios of four algorithms increase with the increasing value of the laxity. This is because the large deadline leads to long slack times, which in turn tend to make the deadlines more likely to be guaranteed.

Figure 8.3b plots security values of the four alternatives when the deadline base is increased from 50 to 800 Sec. Comparing with the average execution time of all jobs in the trace, which is 8030.8 Sec., the laxity range [50, 800] is reasonable. Figure 8.3b reveals that SAREG consistently performs better, with respect to quality of security, than all the other three approaches. When the deadlines are tight, the security values of SAREG are much higher than those of MINMIN and SUFFER. In addition, SAREG consistently outperforms NS-EDF when the laxity varies from 50 seconds to 800 seconds. This is because that SAREG can improve accepted jobs' security levels under constraints of their deadlines and resources availability, while NS-EDF makes no effort to optimize the security levels. More specifically, NS-EDF merely randomly chooses a security level within the corresponding security requirement range. Interestingly, when the deadlines become tight, the performance improvements of SAREG over the three competitor algorithms are more pronounced. The results clearly indicate that Grids can gain more performance benefits from our SAREG approach under the circumstance that real-time tasks have urgent deadlines.

The overall system performance improvements achieved by SAREG are plotted in Figure 8.3c. The first observation deduced from Figure 8.3c is that the value of overall system performance increases with the laxity. This is mainly because the overall system

performance is a product of security value and guarantee ratio, which become higher when the deadlines are loose due to the high laxity value.

A second observation made from Figure 8.3c is that the SAREG algorithm significantly outperforms all three alternatives. This can be explained by the fact that although the guarantee ratios of SAREG and NS-EDF are similar, SAREG considerably improves security values over the other algorithms, while achieving much higher guarantee ratio than MINMIN and SUFFER. This result suggests that if quality of security is the sole objective in scheduling, SAREG is more suitable for Grids than the other algorithms. By contrast, if schedulability is the only performance objective, SAREG can maintain similar guarantee ratios as those of NS-EDF, whose security performance is the second best among the four algorithms.

Last but not least, Figure 8.3c indicates that the overall performance improvement of SAREG over the other three algorithms becomes more pronounced when the deadlines are tighter, implying that more performance benefits can be obtained by SAREG for real-time applications with small slack times. This is because the SAREG approach is less sensitive to the change in deadlines than the other approaches.

8.4.3 Impact of the Number of Nodes

This subsection is focused on performance impact of the number of nodes in a four-site simulated Grid. Specifically, we evaluate the performance of the four algorithms in the cases where the total number of computation nodes in a Grid changes from 184 to 384 and all tasks have very tight deadlines (laxity=50 seconds). Each task in the trace poses requirement on how many nodes it needs. The goal is to examine the performance impact of number of total nodes in a Grid.

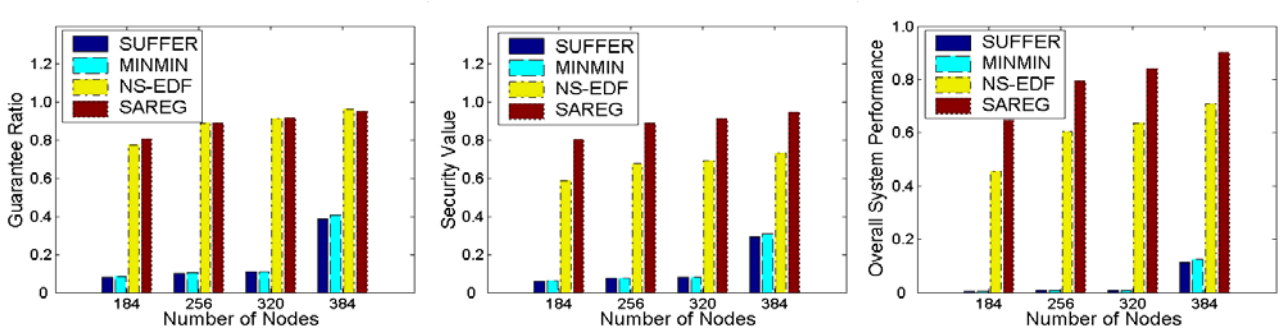


Figure 8.4 Performance impact of number of nodes

Figure 8.4 shows the performance impacts of the number of nodes in the Grid. We observe from the figure that SAREG delivers better overall system performance than the other competitor algorithms. This result is consistent with that observed from the previous experiment (see Figure 8.3). Furthermore, all the four scheduling algorithms exhibit better performance when the Grid has more computation nodes. However, MINMIN and SUFFER can only marginally improve performance in guarantee ratio and security value when more computational nodes are available in the Grid. The reason is two-fold: (1) each task has an extremely tight deadline and the guarantee ratios of MINMIN and SUFFER largely depend on deadlines (see Figure 8.3), and (2) the number of nodes increased is very small compared with the number of total submitted jobs.

Interestingly, Figure 8.4 reveals that the performance improvement of *SAREG* over *NS-EDF* in terms of *GR* is not promising in the first three nodes number configurations. *NS-EDF* even outperforms *SAREG* in *GR* when the total number of nodes in the Grid is 384. The rationale behind this result is that *SAREG* always tries to promote currently arrived tasks' security levels to the maximal possible value, which in turn increases the execution time of currently scheduled tasks. Therefore, subsequent tasks could wait for a

longer time to be executed and thus violate their deadlines. For *NS-EDF* this situation does not apply.

8.4.4 Sensitivities to CPU Capacity

To examine performance sensitivities of the four algorithms to CPU capacity, in this set of experiments we varied the CPU capacity (measured as speedup over the baseline computational node) from 2 to 16. Specifically, the CPU speed of the IBM SP2 66MHz nodes is normalized to 1. We escalate the CPU capacity of the nodes to a normalized value of 2, 4, 8, and 16, respectively. Therefore, the execution times (including security overhead) could be 1/2, 1/4, 1/8 and 1/16 of that of original values, respectively. Also, we select a 200 seconds laxity and a four-site simulated Grid with total 184 nodes. This experiment is focused on evaluating the performance impact of CPU speedup on the four algorithms under a situation where deadlines are relatively tight and the number of nodes is less than sufficient.

The results reported in Figure 8.5 reveal that the SAREG algorithm outperforms the other three alternatives in terms of security value and overall system performance. However, the difference in guarantee ratio performance between SAREG and NS-EDF is almost zero. This is because SAREG can accept the same number of submitted tasks as NS-EDF when the node's CPU speed is so fast that the security overhead is trivial and thus has little effect on the guarantee ratio performance. Figure 8.5 shows that MINMIN and SUFFER only slightly improve their performance when the computing capacity of the Grid is increased. The results can be explained by the following two reasons. First, the trace used in the simulation has approximately a fixed job arrival rate, meaning that decreasing jobs' execution time does not necessarily improve guarantee ratio. Second, the

deadlines of all submitted tasks are relatively tight. As we can see from Figure 8.5, the laxity has great influence on MINMIN and SUFFER in terms of the guarantee ratio.

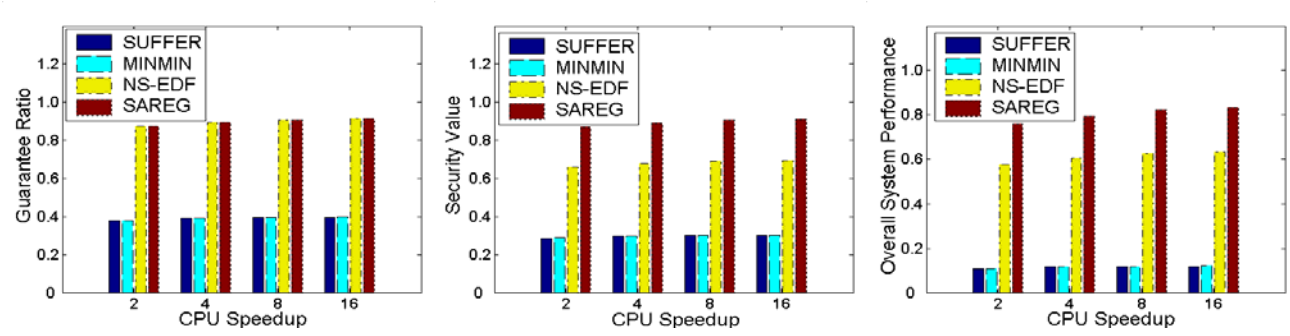


Figure 8.5 Performance impact of CPU Speedup

8.4.5 Scalability

This experiment is intended to investigate the scalability of the SAREG algorithm. We scale the number of sites in the Grid from 4 to 32. Figure 8.6 plots the performances as functions of the number of sites in the Grid. The results show that the SAREG approach exhibits good scalability.

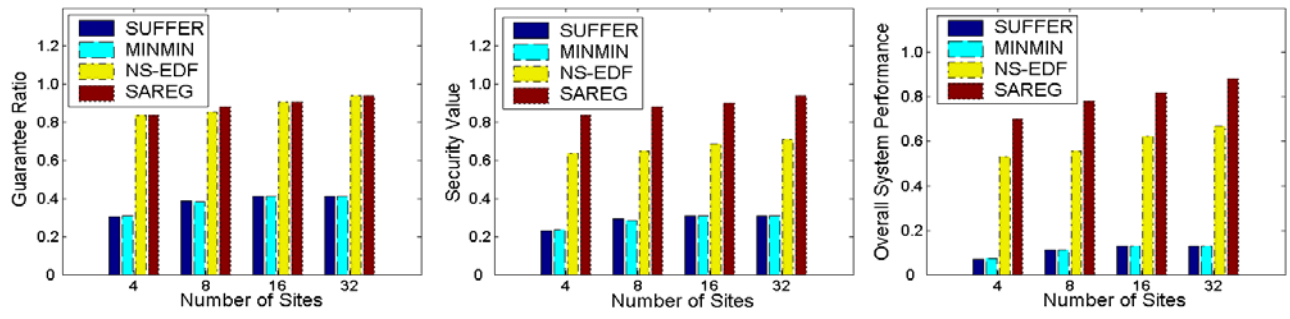


Figure 8.6 Performance impact of number of sites

Figure 8.6 shows the improvement of SAREG in overall system performance over the other three heuristics. It is observed from Figure 8.6 that the amount of improvement over MINMIN and SUFFER maintains almost the same level with the increasing value of

the site number. This result can be explained by the non-real-time nature of MINMIN and SUFFER, which schedules tasks that change the expected site ready time status by the least amount that any assignment could [52].

8.4.6 Conventional performance metrics

In this subsection we compare SAREG with the other three alternatives in terms of conventional performance metrics, namely, mean slowdown and mean response time. The purpose of the comparison is to verify if SAREG has good performance in the two commonly used metrics.

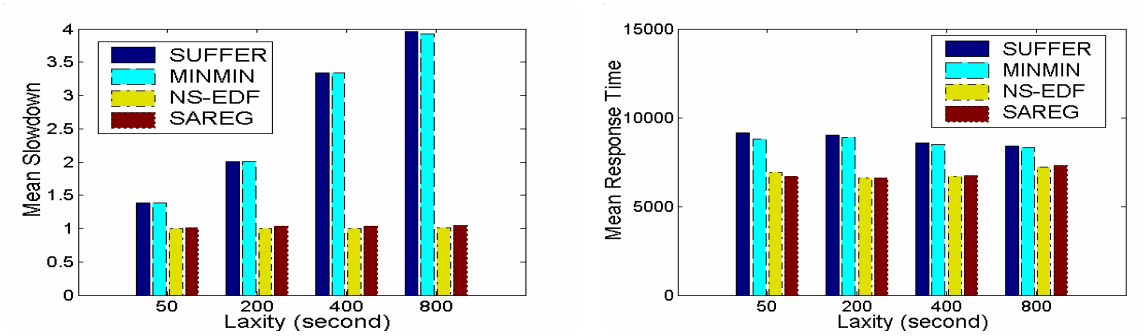


Figure 8.7 Deadline impact on mean slowdown and mean response time

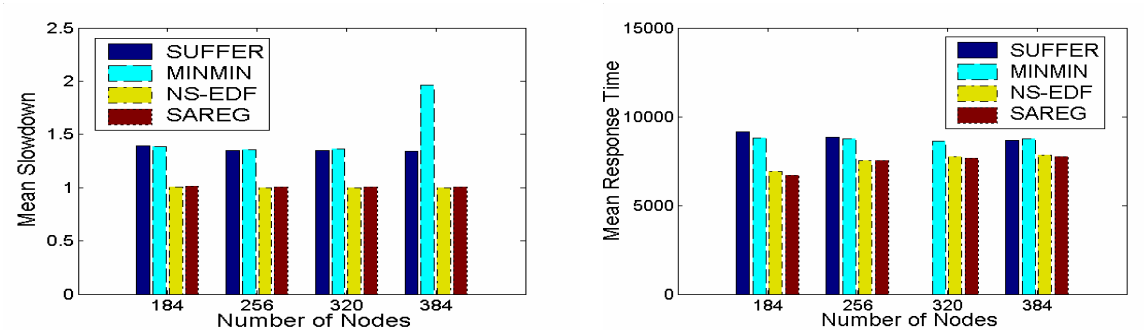


Figure 8.8 Number of node impact on mean slowdown and mean response time

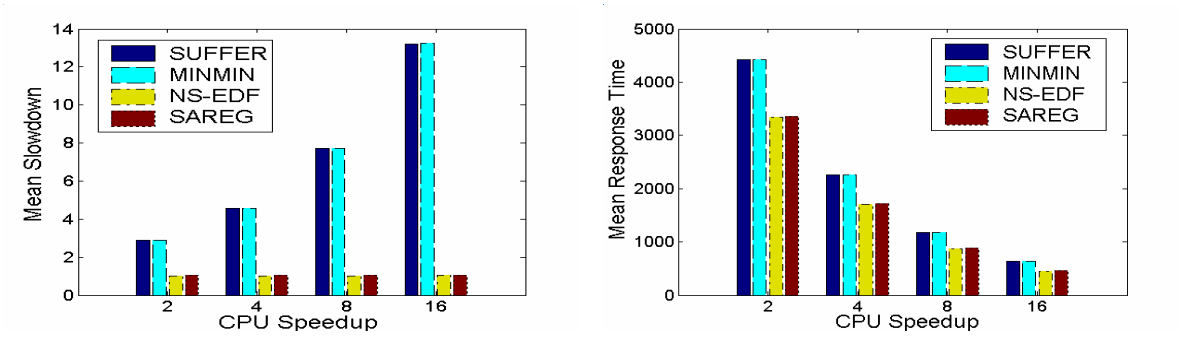


Figure 8.9 CPU speedup impact on mean slowdown and mean response time

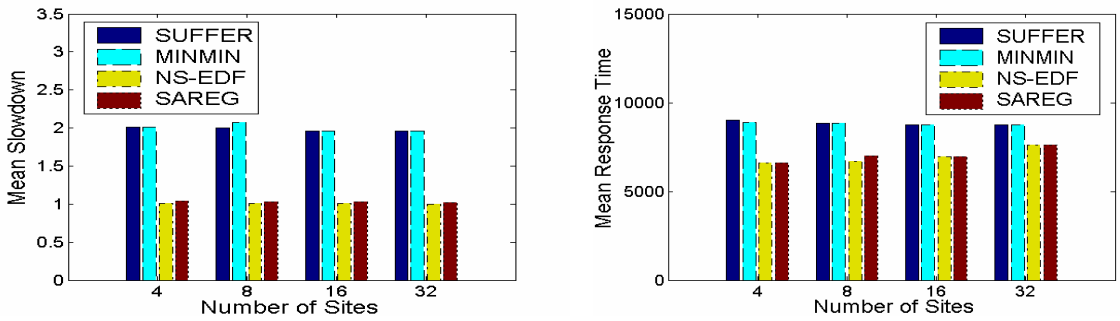


Figure 8.10 Number of site impact on mean slowdown and mean response time

Figures 8.7-8.10 show that SAREG substantially outperforms MINMIN and SUFFER in all four cases described from section 8.4.2 to 8.4.5. It is interesting to observe from Figure 8.9 that the mean response time is very sensitive to the CPU speedup. This is mainly because the execution time of each job is greatly reduced due to the increase in the CPU capacity. SAREG tied with NS-EDF in terms of mean slowdown and mean response time. However, the SAREG algorithm significantly outperforms NS-EDF in security value, which is one of the most important performance metrics in security-sensitive real-time Grid environments. An insightful conclusion we draw from this set of experiments is that SAREG significantly improves the performance in security over

conventional real-time scheduling algorithms like EDF while maintaining a similar level of performance in traditional metrics, including mean slowdown and mean response time.

8.5 Summary

In this chapter, we proposed a novel scheduling algorithm, or *SAREG,* for real-time applications on computational Grids. The SAREG approach paves the way to the design of security-aware real-time scheduling algorithms for Grid computing environments. To make the SAREG scheduling algorithm practical, we presented a mathematical model in which a scheduling framework and security-sensitive real-time jobs are formally described.

To quantitatively evaluate the effectiveness of the SAREG algorithm, we conducted extensive simulations based on a real world trace from a supercomputing centre. Experimental results under a wide spectrum of workload conditions show that SAREG significantly enhances quality of security for real-time applications while maintaining high guarantee ratios. Furthermore, SAREG is capable of minimizing the mean slowdown and response time under various workload characteristics. More importantly, SAREG-EDF achieves overall system performance over three existing scheduling algorithms (MIN-MIN, Sufferage, and EDF) by averages of 286.34%, 272.14%, and 33.86%, respectively.

Chapter 9

9 Conclusions and Future Work

In this dissertation, we have developed an array of security-aware scheduling schemes for real-time applications executing in a cluster-computing environment or a computational Grid. This chapter concludes the dissertation by summarizing the major contributions and describing future research directions. The chapter is organized as follows: Section 9.1 highlights the main contributions of the dissertation. In Section 9.2, we focus on some future directions, which are extensions of our past and current research on security-awareness support for real-time applications. Finally, Section 9.3 summarizes the results and their implications.

9.1 Main Contributions

Over the last decade, clusters have become the fastest growing platforms in high-performance computing. More recently, Grids are emerging as next generation computing platforms for large-scale computation and data intensive problems in industry, academic, and government organizations. Security-sensitive real-time applications such as military

aircraft flight control systems have mandatory security requirements in addition to stringent timing constraints. Conventional real-time scheduling algorithms for clusters and Grids, however, either disregard applications' security needs, and thus expose the applications to security threats, or run applications at inferior security levels without optimizing security performance. In recognition that many applications running on clusters and Grids demand both real-time performance and security, in this dissertation research we investigate the problem of scheduling a set of independent real-time tasks or parallel real-time applications with various security requirements. In what follows, we summarize the main contributions of this dissertation study.

9.1.1 A Security Middleware Model for Real-time Applications

We presented a novel security middleware model, SMW, from which a security-sensitive real-time application can exploit a variety of security services to enhance the safety of its execution on Grids [96]. A quality of security control manager (QSCM), a centerpiece of the SMW model, was designed and implemented to achieve a flexible trade-off between overheads caused by security services and system performance, especially under situations where available resources are dynamically changing and insufficient. Importantly, the security middleware model paves the way towards the design of security-aware real-time scheduling algorithms.

9.1.2 A Security Overhead Model

We proposed an effective model that can approximately yet reasonably measure security overheads experienced by tasks with security requirements [97]. In light of the security overhead model, schedulers are enabled to incorporate security overheads into the process of scheduling real-time tasks.

Since security is achieved at the cost of performance degradation, it is critical and fundamental to quantitatively measure overhead caused by various security services. Unfortunately, less attention has been paid to models used to measure security overheads. Security overhead model for each security services in the context of real-time computing remains an open issue. To enforce security in real-time applications while making security-aware scheduling algorithms predictable and practical, we proposed an effective model that is capable of measuring security overheads experienced by tasks with an array of security requirements. With the security overhead model in place, schedulers are aware of security overheads. Particularly, the model can be employed to compute earliest start times and minimal security overheads of real-time tasks running on clusters and Grids. Without loss of generality, we consider three security services widely deployed in real-world systems, namely, encryption, integrity, and authentication.

9.1.3 Security-Aware Scheduling for Homogeneous Clusters

We presented an analysis of security and real-time performance needs of various applications running on homogeneous clusters. In addition, we proposed a security-aware heuristic strategy that can be integrated with existing real-time scheduling policies. After introducing two new performance metrics, we constructed a simulated cluster where the SAEDF algorithm is implemented and evaluated. We made use of the simulated homogenous cluster to evaluate performance of our approach in terms of guarantee ratio, security value, and overall system performance.

9.1.4 Consideration of the Heterogeneity of Resources

Many existing clusters are heterogeneous in terms of resources including but not limited to CPU, memory, and disk storage. Since heterogeneity in resources inevitably

complicates the scheduling issue of real-time tasks, we devised a security and heterogeneity driven scheduling algorithm to improve security of real-time applications on heterogeneous clusters.

Specifically, we extended the security overhead model presented in Chapter 4 to fit the needs of heterogeneous clusters. Two new performance metrics (degree of security deficiency and risk-free probability) were introduced to quantitatively measure quality of security provided by heterogeneous clusters. In addition to security heterogeneities, we considered heterogeneities with respect to computation time.

9.1.5 Supporting for Parallel Applications

A parallel application can be represented by a *Directed Acyclic Graph* (DAG). In an effort to schedule security-sensitive parallel real-time applications, we need to take precedence constraints as well as timing and security requirements into account.

We substantially extended our work in security-aware scheduling for sequential jobs by developing a security-driven task allocation scheme for parallel applications with deadline, security, and task precedence constraints. TAPADS (*Task Allocation for Parallel Applications with Deadline and Security Constraints*), a task allocation scheme for parallel application with deadline and security constraints, was developed to generate optimal allocations that maximize quality of security and the probability of meeting deadlines for parallel applications running on clusters [98]. The proposed TAPADS scheme factors in security and timing correctness in a way that probabilities of being risk free and meeting deadlines are used as two performance objectives for clusters.

9.1.6 Improving Security for Grids

We constructed a security-aware scheduling strategy, or *SAREG*, for real-time applications on computational Grids [99]. Further, we seamlessly integrated SAREG with the QSCM module outlined in Chapter 3. To quantitatively evaluate performance of the SAREG algorithm, we introduced a new performance metric-security value, which was used to measure the quality of security experienced by all real-time jobs whose deadlines can be met.

9.2 Future Work

In the course of designing and developing security-aware scheduling schemes for clusters and Grids, we have found several interesting issues that are worth to be resolved. This section overviews some of these open issues that need further study. In addition, this section presents opportunities for future work by addressing energy-aware scheduling issues in other application domains.

9.2.1 Refining the Security Overhead Model

The security overhead model proposed in Chapter 4 provides an efficient way to quantify security overhead and to differentiate quality of various security mechanisms. However, there is still much space to further improve the model. In particular, we will refine the model in the following three steps. First, we will extend our security overhead model to multi-dimensional computing resources. To this end, we have considered security overhead in terms of computation time, which is only one of the computing resources consumed by security services. Memory, network bandwidth and storage capacities must be addressed in our future study. Second, we will accommodate more security services in

the security overhead model. Besides the three security services discussed, we plan to include authorization and auditing services in the extended model.

Last but not least, we aim to provide an objective way to quantify quality of security exhibited by a wide range of security mechanisms. For example, a security level assigned to a cryptographic algorithm will have to be determined by a combination of the following factors: (1) length of a key used; (2) resources like time and essential equipment that a third-party needs to break the cryptographic algorithm.

9.2.2 Improving Security for Periodic Tasks on Embedded Systems

Embedded systems, ranging from intelligent vehicle highway system [33] and hearing aids [25] to satellite [47], have been applied to diverse environments, including real-time computing platforms, which depend not only on results of computation, but also on time instants at which these results become available.

Since many embedded systems need to access, store, and manipulate security sensitive data [69], improving quality of security in embedded systems is increasingly becoming a critical and challenging issue in the design and development of embedded, real-time systems. Although there exists a large body of research related to security in the context of general-purpose computing systems, security techniques developed for PCs or servers are inadequate for embedded systems. Securing real-time embedded systems relies on a careful selection of security strategies, which are, in most cases, computational intensive and likely to push computing resources to the limit.

Today there are a variety of systems that have real time and security considerations, because sensitive data and processing require special safeguard and protection against unauthorized access [67]. In particular, real-time applications running in embedded

systems require security protections to completely fulfill their trustworthy computing needs. However, conventional real time systems, which are developed to guarantee timing constraints while possibly posing unacceptable security risks, fail to meet the requirements of information security and assurance for modern real-time applications.

We will investigate fundamental and innovative real-time scheduling algorithms that are intended to achieve high quality of security for embedded systems while improving resource utilization.

9.2.3 Energy-Saving for Embedded Systems

Parallel applications with energy and low-latency constraints are emerging in various networked embedded systems like digital signal processing, vehicle tracking, and infrastructure monitoring. However, conventional energy-driven task allocation schemes for a cluster of embedded nodes only concentrate on energy-saving when making allocation decisions. Consequently, the length of the schedules could be very long, which is unfavorable or in some situations even not tolerated.

We will address the issue of allocating a group of parallel tasks on a heterogeneous embedded system with an objective of energy-saving and short-latency. A novel task allocation strategy needs to be developed to find an optimal allocation that minimizes overall energy consumption while confining the length of schedule to an ideal range.

9.3 Conclusions

This dissertation has presented a family of security-aware scheduling techniques in cluster and Grid computing environments. These techniques pave a way to the design of security-aware real-time systems. To make the security-aware scheduling strategies

delineated in this dissertation practical, we proposed a security overhead model to quantitatively measure overheads of security services including but not limited to confidentiality, integrity, and authentication required by real-time applications. In doing so, security overheads can be taken into consideration in the process of scheduling real-time tasks. The effectiveness of our strategies was evaluated by developing an array of novel security-aware real-time scheduling schemes, which were seamlessly integrated with existing real-time scheduling policies like EDF (Earliest Deadline First). To quantitatively validate the performance of our security-aware schemes, we first introduced some new performance metrics, such as, security value (see Equation 5.4). Security value is a collective value of each admitted application's security level and it can be used to measure the quality of security experienced by all schedulable real-time tasks. We then conducted extensive trace-driven simulations using synthetic and real-world traces in addition to real-world applications. and Experimental results show that our security-aware strategies outperform existing scheduling algorithms in terms of security value while consistently maintaining a high performance in conventional metrics like guarantee ratios under a wide range of workload characteristics.

Bibliography

[1] T.F. Abdelzaher and K.G. Shin., "Combined Task and Message Scheduling in Distributed Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* Vol. 10, No. 11, Nov. 1999.

[2] T.F. Abdelzaher, E. M. Atkins, and K.G. Shin., "QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control," *IEEE Trans. Computers,* 49(11), Nov. 2000, pp.1170-1183.

[3] I. Ahmad, and A. Ghafoor, "Semi-distributed load balancing for massively parallel multi computer systems," *IEEE Trans. Software Eng*. pp. 987-1004, vol. 10, 1991.

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

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

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

[7] F. Azzedin, and M. Maheswaran, "Towards trust-aware resource management in grid computing systems," *Proc. 2nd IEEE/ACM Int'l Symp. Cluster Computing and the Grid,* May 2002.

[8] R. Bajaj and D.P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, Issue 2, pp. 107-118, 2004.

[9] Jorge Barbosa, C. Morais, R. Nobrega, and A.P. Monteiro, "Static scheduling of dependent parallel tasks on heterogeneous clusters," *IEEE International Conference on Cluster Computing*, Boston, Massachusetts, USA, September 27 - 30, 2005.

[10] P.A. Bernstein, "Middleware: A Model for Distributed System Services," *Communication of ACM* 39(2), pp. 86-98, 1996.

[11] V. Berten, J. Goossens, and E. Jeannot, "On the Distribution of Sequential Jobs in Random Brokering for Heterogeneous Computational Grids," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17, Issue 2, pp. 113-124, Feb. 2006.

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

[13] N.J. Boden, D. Cohen, and W.K. Su., "Myrinet: A Gigabit-per-second Local Area Network," IEEE Micro, Vo. 15, No. 1, February 1995.

[14] C. Boeres, and V. Rebello, "Cluster-Based Static Scheduling: Theory and Practice," Proc. 14th Symposium on Computer Architecture and High Performance Computing, pp.133-140, 2002.

[15] A. Bosselaers, R. Govaerts, and J. Vandewalle, "Fast hashing on the Pentium," *Proc. Advances in Cryptology*, LNCS 1109, pp. 298-312, Springer-Verlag, 1996.

[16] T. D. Braun et al., "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," *Proc. Workshop on Heterogeneous Computing*, pp.15-29, Apr. 1999.

[17] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *Technical Report CS-94-01*, University of Virginia, Computer Science Department, January 1994.

[18] A.J. Chakravarti, G. Baumgartner, and M. Lauria, "Application-specific scheduling for the organic grid," *Proc. Fifth IEEE/ACM Inte'l Workshop on Grid Computing*, 2004.

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

[20] A.T. Chronopoulos, S. Penmatsa, and Ning Yu, "Scalable loop self-scheduling schemes for heterogeneous clusters," *Proc. Int'l Conf. Cluster Computing*, pp. 353-359, Sept. 2002.

[21] W.W. Chu, M.T. Lan, and J. Hellerstein, "Estimation of inter module communication (IMC) and its applications in distributed processing systems," *IEEE Trans. on Computers*, pp.691-299, Aug., C-33, 1984.

[22] K. Connelly and A. A. Chien, "Breaking the barriers: high performance security for high performance computing," *Proc. Workshop on New security paradigms*, Virginia, Sept. 2002.

[23] J. Deepakumara, H.M. Heys, and R. Venkatesan, "Performance comparison of message authentication code (MAC) algorithms for Internet protocol security (IPSEC)," *Proc. Newfoundland Electrical and Computer Engineering Conf.*, St. John's, Newfoundland, 2003.

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

[25] L.P.L. van Dijk, A. C. van der Woerd, J. Mulder, A. H. M. van Roermund, "An Ultra-Low-Power, Low-Voltage Electronic Audio Delay Line for Use in Hearing Aids," *IEEE Journal of Solid-State Circuits*, Vol. 33, No. 2, pp. 291-294, Feb. 1998.

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

[27] E. Durant, "Embedded Real-Time System Considerations," *EECS Department of Milwaukee School of Engineering*, April 23, 1998.

[28] O. Elkeelany, M. Matalgah, K. Sheikh, M. Thaker, G. Chaudhry, D. Medhi, J. Qaddouri, "Performance analysis of IPSec protocol: encryption and authentication," *Proc. IEEE Int'l Conf. Communications*, pp. 1164-1168, New York, NY, April-May 2002.

[29] M. Eltayeb, A. Dogan, and F. Ozunger, "A data scheduling algorithm for autonomous distributed real-time applications in grid computing," *Proc. Int'l Conf. Parallel Processing*, Aug. 2004.

[30] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow,J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous computing environments with SmartNet," *Proc. Heterogeneous Computing Workshop*, pp.184-199, March 1998.

[31] D. Gajski and J. Pier, "Essential issues in multiprocessors," *IEEE Computer*, vol.18, June, 1985.

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

[33] D. N. Godbole, J. Lygeros, and S. S. Sastry, "Hierarchical Hybrid Control: an IVHS Case Study," Proc. Conf. Control and Decision, pp. 1592-1597, 1994.

[34] T. Hagras and J. Janecek, "A static task scheduling heuristic for homogeneous computing environments," *Proc. 12th Euromicro Conf. Parallel, Distributed and Network-Based Processing*, pp.192-198, 2004.

[35] W. A. Halang, et al., "Measuring the performance of real-time systems," *Int'l Journal of Time-Critical Computing Systems*, 18, pp. 59-68, 2000.

[36] B. Hamidzadeh and D.J. Lilja, "Dynamic scheduling strategies for shared-memory multiprocessors," *Proc. 16th Int'l Conf. Distributed Computing Systems*, pp. 208-215, 1996.

[37] A. Harbitter and D. A. Menasce, "The performance of public key enabled Kerberos authentication in mobile computing applications," *Proc. ACM Conf. Computer and Comm. Security*, 2001.

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

[39] L. He, A. Jatvis, and D. P. Spooner, "Dynamic scheduling of parallel real-time jobs by modelling spare capabilities in heterogeneous clusters," *Proc. Int'l Conf. Cluster Computing*, pp. 2-10, Dec. 2003.

[40] C.-J. Hou and K. G. Shin, "Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems," *IEEE Trans. Computers*, Vol. 46, No. 12, Dec. 1997.

[41] J. Huang, Y. Wang and F. Cao, "On Developing Distributed Middleware Services for QoS- and Criticality-Based Resource Negotiation and Adaptation," *Real-Time Systems* 16(2): 187-221; May 1999.

[42] M. Humphrey, M. R. Thompson, and K. R. Jackson, "Security for Grids," *Proc. of the IEEE*, pp. 644-652, March 2005.

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

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

[45] V. Kalogeraki, P.M. Melliar-Smith, and L.E. Moser, "Dynamic scheduling of distributed method invocations", *Proc. 21st Real-Time Systems Symposium*, pp.57 – 66, 2000

[46] A. Karageorgos and H. Karatza, "Performance issues of task routing and task scheduling with resequencing in homogeneous distributed systems," *Proc. 30th Annual Simulation Symp.*, pp. 56-63, 1997.

[47] K. Kawano, J. Kudoh, "Real-Time Satellite Data Transfer System for Siberian NOAA Image Database," *Proc. Int'l Symp. Geoscience and remote Sensing*, Vol. 5, pp. 2277-2279, July 2001.

[48] 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.

[49] J. Lee, B. Tierney, and W. Johnston, "Data intensive distributed computing: a medical application example," *Proc. High Performance Computing and Networking Conf.*, April 1999.

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

[51] C. L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, Vol.20, No.1, pp. 46-61, 1973.

[52] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamicmatching and scheduling of a class of independent tasks onto heterogeneous comput-ing systems," Proc. *IEEE Heterogeneous Computing Workshop*, pp. 30–44, Apr. 1999.

[53] M. Maheswaran and H.J. Siegel, "A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. the Seventh Heterogeneous Computing Workshop*, pp.57-69, 1998.

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

[55] K. Nahrstedt, D. Xu, D. Wichadakul, B. Li, "QoS-aware middleware for ubiquitous computing," *IEEE Communications Magazine*, 39(11):140--148, November 2001.

[56] E. Nahum, S. O'Malley, H. Orman, R. Schroeppel, "Towards High Performance Cryptographic Software," *Proc. IEEE Workshop Architecture and Implementation of High Performance Communication Subsystems*, August 1995.

[57] J. Nilsson and F. Dahlgren, "Improving performance of load-store sequences for transaction processing workloads on multiprocessors," *Proc. Int'l Conference on Parallel Processing*, pp. 246-255, 21-24 Sept. 1999.

[58] A.J. Page, and T.J. Naughton, "Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing," *The 19th IEEE Int'l Parallel and Distributed Processing Symposium*, Denver, USA, 2005.

[59] M. Palis, A. Liou, J.C. Rajasekaran, S. Shende, and D. Wei, "Online scheduling of dynamic trees," *Parallel Processing Letter*, pp. 635-646, Dec., 1995.

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

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

[62] X. Qin and H. Jiang, "Improving Effective Bandwidth of Networks on Clusters using Load Balancing for Communication-Intensive Applications," *Proc. 24th IEEE Int'l Performance, Computing, and Communications Conf.,* Phoenix, Arizona, April 2005.

[63] X. Qin and H. Jiang, "Data Grids: Supporting Data-Intensive Applications in Wide Area Networks," *High Performance Computing: Paradigm and Infrastructure*, edited by L. T. Yang and M. Guo, John Wiley and Sons, 2004.

[64] X. Qin, H. Jiang, and 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.

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

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

[67] K. Ramamritham and J. A. Stankovic, "Dynamic task scheduling in distributed hard real-time system," *IEEE Software,* Vol. 1, No. 3, July 1984.

[68] 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.

[69] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in Embedded Systems: Design Challenges," *ACM Trans. Embedded Computing Systems*, Vol. 3, No. 3, pp. 461-491, Aug 2004.

[70] R.M. Santos, J. Santos, and J. Orozco, "Scheduling heterogeneous multimedia servers: different QoS for hard, soft and non real-time clients," *The 12th Euromicro Conf. Real-Time Systems*, pp. 247-253, 2000.

[71] B. Schneier, "Authentication and Expiration," *IEEE Security and Privacy Magazine*, Vol. 3, No. 1, 2005.

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

[73] 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.

[74] S. H. Son, R. Zimmerman, and J. Hansson, “An adaptable security manager for real-time transactions,” *Proc. 12th Euromicro Conf. Real-Time Systems*, pp. 63 – 70, June 2000.

[75] S. H. Son, R. Mukkamala, and R. David, “Integrating security and real-time requirements using covert channel capacity,” *IEEE Trans. Knowledge and Data Engineering*, Vol. 12 , No. 6, pp. 865 – 879, 2000.

[76] 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.

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

[78] T. L. Sterling, J. Salmon, D. Becker, and D. Savarese, “How to Build a Beowulf. A Guide to the Implementation and Application of PC Clusters,” The MIT Press, 1999.

[79] V. Subramani, R. Kettimuthu, S. Srinivasan, and S. Sadayappan, “Distributed job scheduling on computational Grids using multiple simultaneous requests,” *11th IEEE International Symposium on High Performance Distributed Computing*, July 2002.

[80] M. Tan, H.J. Siegel, J.K. Antonio, and Y.A. Li, “Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, Issue 8, pp. 857-871, 1997.

[81] A.S. Tanenbaum, M. Steen, “Distributed Systems: Principles and Paradigms”, ISBN 0130888931, *Prentice Hall, 1st edition,* January 15, 2002.

[82] M. D. Theys, M. Tan, N. Beck, H. J. Siegel, and M. Jurczyk, “A mathematical model and scheduling heuristic for satisfying prioritized data requests in an oversubscribed communication networks,” *IEEE Trans. Parallel and Distributed Systems*, Vol. 11, No. 9, pp. 969-988, Oct. 2000.

[83] M.E. Thomadakis and J.-C. Liu, "On the efficient scheduling of non-periodic tasks in hard real-time systems," *Proc. 20th IEEE Real-Time Systems Symp.*, pp.148-151, 1999.

[84] 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.

[85] G. Vallee, C. Morin, J.-Y. Berthou, and L. Rilling, "A new approach to configurable dynamic scheduling in clusters based on single system image technologies," *Proc. Int'l Symp. Parallel and Distributed Processing*, April 2003.

[86] S. Vazhkudai , J. M. Schopf , and I. Foster, "Predicting the Performance of Wide Area Data Transfers," *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, April 2002.

[87] S. Viswanathan,B. Veeravalli, D. Yu, and T.G. Robertazzi, "Design and analysis of a dynamic scheduling strategy with resource estimation for large-scale grid systems," *Proc. 5th IEEE/ACM Int'l Workshop on Grid Computing*, pp. 163-170, 2004.

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

[89] S.D. Wang, I.T. Hsu, and Z.Y. Huang, "Dynamic scheduling methods for computational grid environments," *Proc. 11th Int'l Conf. Parallel and Distributed Systems*, Vol.1 , pp. 20-28, 2005.

[90] C. Wang, W.A. Wulf, "Towards a Framework for Security Measurement," *Proc. National Information Systems Security Conference*, Baltimore, MD, pp. 522-533, October, 1997.

[91] B. Wilkinson and M. Allen, "Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers", *Prentice-Hall, Inc.*, 1999.

[92] R. Wolski, "Experiences with Predicting Resource Performance On-line in Computational Grid Settings," *ACM SIGMETRICS Performance Evaluation Review*, Volume 30, Number 4, pp 41--49, March, 2003.

[93] C.M. Woodside and G.G. Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems", IEEE Trans. on Parallel and Distributed Systems, Vol. 4, No. 2, pp. 164-174, Feb. 1993.

[94] J. Wray, RFC2744- Generic Security Service API Version 2: C-bindings, http://www.faqs.org/rfcs/rfc2744.html, January, 2000.

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

[96] T. Xie and X. Qin, "A Security Middleware Model for Real-time Applications on Grids," *IEICE Transactions on Information and Systems, Special Issue on Parallel/Distributed Computing and Networking*, February 2006

[97] T. Xie and X. Qin, "Scheduling Security-Critical Real-Time Applications on Clusters," *IEEE Transactions on Computers* (Accepted, regular paper), Jan. 2006.

[98] T. Xie and X. Qin, "A New Allocation Scheme for Parallel Applications with Deadline and Security Constraints on Clusters," *Proc. IEEE Int'l Conf. Cluster Computing*, September 27-30, Boston, USA, 2005.

[99] 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*, pp.146-158, Cambridge, MA, June 19, 2005.

[100] T. Xie, X. Qin, A. Sung, "SAREC: A Security-Aware Scheduling Strategy for Real-Time Applications on Clusters," *Proc. 34th Int'l Conf. Parallel Processing*, pp.5-12, Norway, June 14-17, 2005.

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

[102] C.C. Yu and S. Bhattacharya, "Dynamic scheduling of real-time messages over an optical network," *Proc. 6th Int'l Conf. Computer Communications and Networks*, pp. 22-25, 1997.

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

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

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

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

[107] Q. Zheng and K.G. Shin, “On the Ability of Establishing Real-Time Channels in Point-to-Point Packet Switched Network,” IEEE Trans. Comm., Vol. 42, 1994.