PAPER Special Section on Parallel/Distributed Computing and Networking

A Security Middleware Model for Real-Time Applications on Grids

Tao XIE[†a)] ***and*** **Xiao QIN**[†], ***Nonmembers***

SUMMARY Real-time applications are indispensable for conducting research and business in government, industry, and academic organizations. Recently, real-time applications with security requirements increasingly emerged in large-scale distributed systems such as Grids. However, the complexities and specialties of diverse security mechanisms dissuade users from employing existing security services for their applications. To effectively tackle this problem, in this paper we propose a security middleware (SMW) model from which security-sensitive real-time applications are enabled to exploit a variety of security services to enhance the trustworthy executions of the applications. A quality of security control manager (QSCM), a centerpiece of the SMW model, has been 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 in distributed systems as large-scale as Grids. Our empirical studies based on real world traces from a supercomputing center demonstratively show that the proposed model can significantly improve the performance of Grids in terms of both security and schedulability.

key words: *security middleware, real-time applications, real-time scheduling, grid*

1. Introduction

An increasing number of real-time systems have timing and security constraints because sensitive data and processing require special safeguards against unauthorized access [4], [5]. In particular, a variety of military real-time applications running on parallel and distributed systems like clusters and Grids require security protections to completely fulfill their security needs. Unfortunately, conventional wisdom on the design of real time systems is inadequate for security-sensitive real-time applications because it did not factor in the applications' security needs.

To tackle the aforementioned problem, we propose a security middleware model, which allows real-time applications to invoke various underlying security services through specific application programming interfaces (APIs) to satisfy their security needs. Employing the security services, however, requires extra overhead in terms of CPU time, network and disk bandwidth. Thus, real-time scheduling algorithms need to consider the overhead to make efficient schedules for tasks submitted. Consequently, applications or users are able to receive satisfactory service from real-time systems, which achieve high performance with respect to quality of security and schedulability. The security middleware model can benefit both applications and the real-time systems. With the model in place, applications or users are allowed to formally describe their security requirements using security services specifications, e.g., security-related APIs. These APIs then invoke an array of high-level security services provided by the framework of the SMW model (see Fig. 1). From a real-time system standpoint, it can leverage the model to glean global information pertinent to the applications' security needs. Additionally, the model makes it possible for the real-time system to measure the applications' security overhead. In doing so, the model is able to make an effort to guarantee timing constraints and security requirements. In a security-critical real-time system, a task will be rejected by the system if the task's minimal security requirements cannot be met. This process is essential because running tasks without guaranteeing their security requirements tends to make the system vulnerable to attack. In short, the model is intended to seamlessly integrate security into real-time scheduling for applications running in parallel and distributed systems.

The contributions of this paper are three-fold. First, a security middleware (SMW) model is proposed. Second, a security-aware real-time scheduling mechanism is implemented. Finally, a case study illustrates the performance of the security-aware real-time scheduling mechanism in the light of the security middleware (SMW) model. Our simulator combines performance and security overhead estimates using the security overhead model based on the three most commonly used security services, i.e., authentication, integrity, and confidentiality. We have used real world traces from a supercomputing centre to drive our simulations. Empirical results demonstrate that the proposed model, in which the scheduling mechanism is the centerpiece, is capable of achieving high quality of security while guaranteeing timing constraints of real-time applications.

The rest of this paper is organized as follows. Related work is discussed in Sect. 2. Section 3 introduces the architecture of our security middleware (SMW) model. Section 4 implements and evaluates the QSCM module, a core part of the model, on a simulated Grid. Section 5 concludes the paper with some comments on future work.

Manuscript received April 4, 2005.
Manuscript revised August 15, 2005.

[†]The authors are with the Department of Computer Science, New Mexico Institute of Mining and Technology, Socorro, NM 87801, USA.
a) E-mail: xietao@cs.nmt.edu

DOI: 10.1093/ietisy/e89–d.2.631

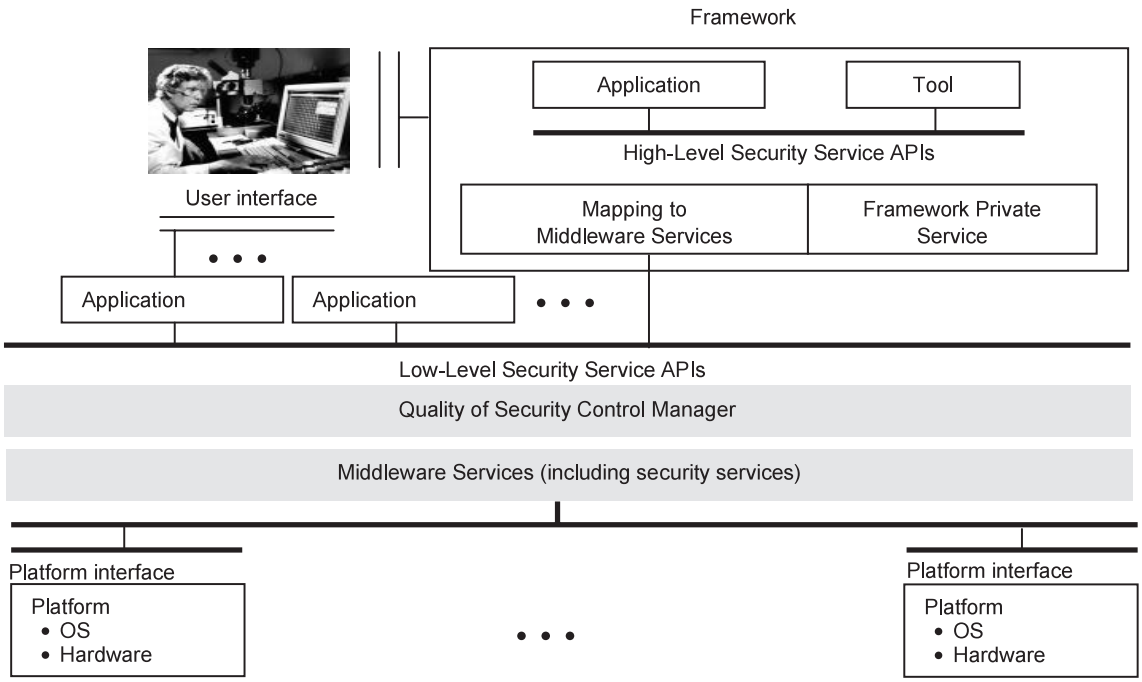


Fig. 1 Security middleware architecture.

2. Related Work

QoS-aware middleware has been extensively studied in the past both experimentally and theoretically [2], [3]. 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 [3]. Abdelzaher et al. presented a scheme for QoS negotiation in real-time applications. The scheme provides a generic way to express application-level semantics to control how application QoS is to be degraded under overload or failure conditions [2]. 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. Our work is orthogonal and complementary to the above approaches in the sense that the security middleware model centered around security services is focused on the security needs of real-time applications.

The security middleware model (SMW) provides a way of explicitly specifying the 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 applications' security requirements 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 [1]. 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 [1]. Wang et al. presented a security measurement framework, which is based on theory and practice of formal measurements [7]. In our previous work [6], we proposed a practical security overhead model to estimate the CPU time overhead of some commonly used security services like authentication and integrity. Our security overhead model leveraged the results in [15], [16], which provided the CPU time units cost for primitive security services such as confidentiality and integrity check. Take confidentiality for example, Nahum et al. in [16] 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. Detailed information about how to quantitatively measure security overhead can be found in our previous work [6], [14]. Most recently, we proposed a family of dynamic security-aware scheduling algorithms for a cluster [6] and a Grid [14].

3. 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 [8]. 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 presents an overview of the archi-

tecture for the SMW model. Detailed functional descriptions of each component of the SMW model can be found in Sect. 3.2. Section 3.3 illustrates how to specify applications' security requirements.

3.1 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 (Fig. 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 his applications with security requirements.

A *framework* is a software environment that is designed to simplify application development and system management for a specialized application domain [8]. The framework illustrated in Fig. 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 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. 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 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 [9], 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 [9]. 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 mechanism.

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 schedulability. Detailed description of QSCM will be given in Sect. 3.2.

A middleware service is a generic service that operates between platforms and applications (see Fig. 1). The middleware service, which is defined by APIs and supported protocols [8], 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, confidentiality 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 in forms of 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 [10].

3.2 Quality of Security Control Manager (QSCM)

QSCM (Fig. 2) is a centerpiece of the SMW model because it can optimize the quality of security services requested by applications while maintaining a high-level system performance in terms of schedulability. 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.

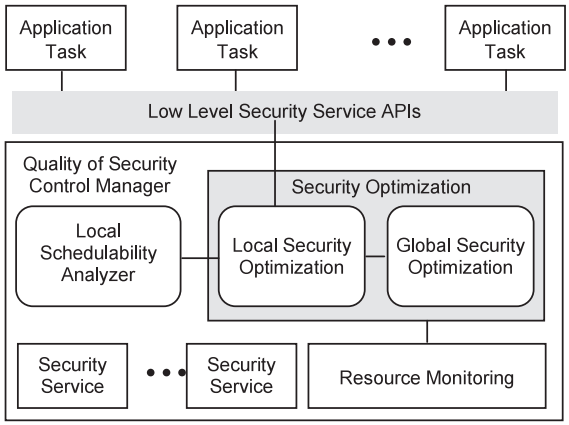


Fig. 2 Quality of security control manager.

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 [7]. 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).

Please note that security mechanisms are not independent of one another. Instead, it is common that multiple security mechanisms are needed in order to form an integrated security solution. For example, authentication and message integrity cannot work without each other [13]. The SMW model offers users an array of basic security mechanisms so that they can select one or multiple of them to form an integrated security solution to meet their security needs. In other words, 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.

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.

The QSCM runs on top of middleware services and uses the Resource Monitoring module to monitor the underlying available resources. A user is enabled to 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 Sect. 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.

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

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 Fig. 1) under the timing constraint:

$$SB(x_i) = \max_{x_i \in X_i} \left\{ \sum_{j=1}^{q} w_i^j s_i^j(x_i) \right\}, \tag{2}$$

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 as 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:

$$SV(x) = \max_{x \in X} \left\{ \sum_{i=1}^{p} y_i SB(x_i) \right\}, \tag{3}$$

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 client's service requests. In addition, the local security optimization module also 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 send a warning message back to the client. This function enforces that only practical security solution requests can be granted.

3.3 Security Service Requirements Specification

In this subsection we present an approach to specifying users' security service requirements. Irvine *et al*. proposed the notion of security range that consists of a set of security levels [11]. Users can define their security requirements for a particular 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 upon the submissions of their tasks. Figure 3 illustrates an example of the task submission structure (TSS) described in TSDL.

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 Fig. 3). After a user submits a TSS, the security service constraints will be translated into the high-level security service APIs. Therefore, there is no need for users to directly deal with the APIs. To further alleviate users' burden, the framework of the SMW model makes it possible for system administrators to specify security requirement expressions using a higher-level abstraction like security abstract table.

```
DEFINE Task : flight_control
{
        Input = (altitude: 1230, heading: 35, …);
        Output = (takeoff_distance, climb_rate);
        Type = "Real Time";
        Period = 80;
        Completion_Time = 0;
        Owner = "sqin";
        Cmd = "flight_con";
        Processor_num= 5;
        Data_secured=250;
        Priority = 3;
        Constraint
            • 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
}
```

Fig. 3 Task submission structure for flight control.

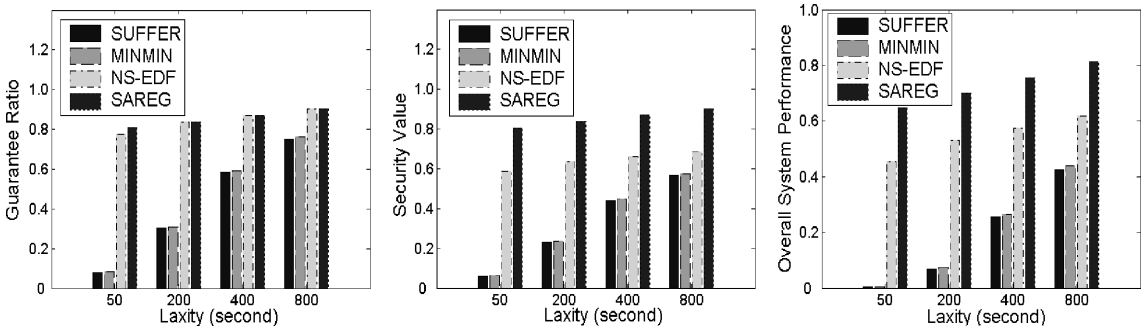
4. QSCM Implementation and Results

The security middleware model (SMW) can take full advantage of the QSCM module (see Fig. 2) to guarantee a diversity of security requirements while improving the schedulability performance of real-time systems. It is intuitive that higher security requirements imply longer execution times, which in turn may violate real-time tasks' deadlines. In the light of the schedulability analyzer and security optimization module, the scheduling mechanism in the QSCM module is capable of making appropriate trade-offs between security and real-time requirements. To quantitatively evaluate the performance of the proposed model, we implemented the QSCM module based on the model presented in Sect. 3. In particular, we implemented the SAREG scheduling algorithm [14] in the QSCM module. Note that SAREG detailed in [14] is a real-time scheduling algorithm with security-awareness.

Experimental results demonstrate that the performance gain of the QSCM module, in which the SAREG algorithm is employed, is significant due to the virtue of the security middleware model. Therefore, the SMW model can eventually enhance the performance of distributed real-time systems in terms of security and schedulability (measured as guarantee ratio). Section 4.1 presents an introduction of the simulation experiments. Section 4.2 compares SAREG with the three baseline algorithms. Section 4.3 shows that SAREG still maintains a good performance in conventional performance metrics.

4.1 Introduction of Experiments

Using extensive simulation experiments based on San Diego Supercomputer Center (SDSC) SP2 log, we evaluate in this section the potential benefits of the proposed security middleware model. We constructed a homogeneous Grid [14] simulator, which assumes that (1) there are multiple sites in the Grid; (2) the number of nodes in one site could be different from the number of nodes in another site; and (3) all nodes in the simulated Grid had an identical processing power. The last assumption is reasonable in the sense that it can be easily relaxed by incorporating a simple conversion mechanism for relative heterogeneous processing capabilities. Therefore, it's readily to extend our SAREG scheme to a heterogeneous Grid. Besides, to provide workload (arrival tasks) for each site in the simulated Grid, we randomly divided the trace into multiple parts and each part went to one site. Similar experimental methodology was used in [17], which collected real workload from one computer in six daytime intervals. And then a distributed system, where each of six hosts executes the workload arrivals from one of the daytime traces, was simulated [17]. In addition, using



(a) Guarantee ratio. (b) Security value. (c) Overall system performance.

Fig. 4 Performance impact of deadline.

trace from one supercomputing center to drive a simulated Grid is reasonable because (1) a Grid is essentially a large-scale virtual super-site that consists of multiple sites; (2) the workload of each site in our simulated Grid came from partial workload of one existing real site (SDSC).

In purpose of revealing the strength of the QSCM module with SAREG, we compared it against two scenarios where QSCM make use of three well-know scheduling algorithms, namely, Min-Min, Sufferage [12], and earliest deadline first algorithm (EDF). To highlight the non-security-aware characteristic of the EDF algorithm, we call it NS-EDF (non-security-aware EDF) in this paper. Although NS-EDF algorithm, a slightly modified version of EDF, is intended to schedule real-time tasks with security requirements, it makes no effort to optimize quality of security. For the sake of simplicity, throughout this section Sufferage is referred to SUFFER. The three baseline algorithms are briefly described below.

(1) MINMIN: For each submitted task, the node 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 node (machine).

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

(3) NS-EDF: The task with the earliest deadline is always executed first.

The admission controller randomly selects a security level of each security service required by an arrival task for all the three baseline algorithms above. The purpose of comparing SAREG with MINMIN and SUFFER is to show the performance improvements over existing Grid scheduling algorithms in a real-time computing environment where the QSCM module were not deployed. The goal of comparing SAREG with NS-EDF is to demonstrate the security performance benefits gained by integrating SAREG in the QSCM module. The performance metrics by which we evaluate system performance include: *security value* (the total sum of all accepted tasks' security levels), *guarantee ratio* (measured as a fraction of total submitted tasks that are found to be schedulable), *overall system performance* (defined as a product of security value and guarantee ratio).

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).

Figure 4 shows the simulation results for these four algorithms on a four-site Grid with 184 nodes where the CPU power is fixed at 100MIPS. We observe from Fig. 4 (a) that SAREG and NS-EDF exhibit similar performance in terms of guarantee ratio (the performance difference is less than 2%), whereas SAREG noticeably outperforms MINMIN and SUFFER algorithms. Figure 4 (b) plots security values of the four algorithms when the deadline base is increased from 50 second to 800 second.

Figure 4 (b) 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 larger than that of MINMIN and SUFFER. In addition, SAREG also consistently outperforms NS-EDF. This is because that SAREG can promote all accepted tasks' security levels under constraints of their deadlines and resources availability, while NS-EDF puts no effort into optimizing submitted tasks' security level at all.

4.3 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.

Figures 5 and 6 shows us that SAREG substantially outperforms MINMIN and SUFFER. SAREG tied with NS-EDF in terms of mean slowdown and mean response time. However, SAREG greatly outperforms NS-EDF in security value, which is one of the most important performance metrics in a security-critical real-time Gird computing environment.

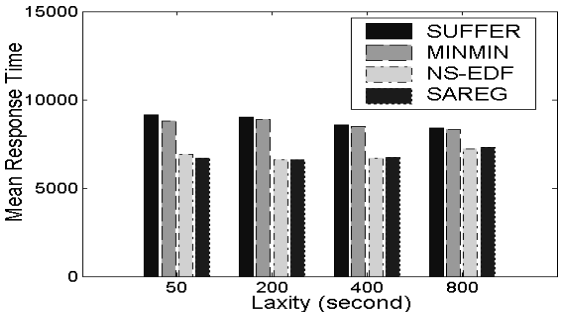


Fig. 5 Deadline impact on mean response time.

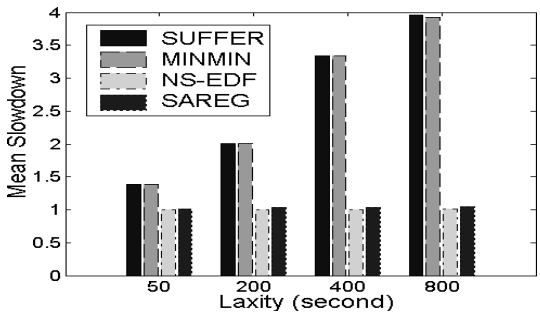


Fig. 6 Deadline impact on mean slowdown.

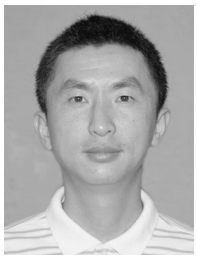
5. Summary and Future Work

In this paper, 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 Grids. In addition, we constructed a security-aware scheduling strategy, or SAREG, for real-time applications on Grids by integrating the QSCM module into the scheduling mechanism. This strategy paves a way to the design of security-aware real-time scheduling algorithms. The effectiveness of the SAREG strategy was evaluated by developing a new security-aware real-time scheduling algorithm (SAREG), which incorporates the earliest deadline first (EDF) scheduling algorithm into the SAREG strategy. To quantitatively validate the performance of our SAREG algorithm, we conducted trace-driven simulations and introduced two new performance metrics, namely, *security value* and *overall system performance*. Simulation results on various simulated Grids show that SAREG achieves overall system performance over three baseline real-time scheduling algorithms (MINMIN, SUFFER and NS-EDF) by averages of 286.34%, 272.14%, and 33.86%, respectively. In addition, the empirical results reveal that SAREG significantly improves quality of security for real-time tasks while maintaining high guarantee ratios under a wide range of workload characteristics.

Future studies in this research can be performed in the following directions. (1) 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. (2) Accommodate more security services like authorization and auditing services into consideration into our SMW model.

References

[1] C. Irvine and T. Levin, "Towards a taxonomy and costing method for security services," Proc. 15th Annual Computer Security Applications Conference, pp.183–188, 1999.
[2] T.F. Abdelzaher, E.M. Atkins, and K. Shin, "QoS negotiation in real-time systems and its application to automated flight control," IEEE Trans. Comput., vol.49, no.11, pp.1170–1183, Nov. 2000.
[3] J. Huang, Y. Wang, and F. Cao, "On developing distributed middleware services for QoS- and criticality-based resource negotiation and adaptation," Real-Time Systems, vol.16, no.2, pp.187–221, May 1999.
[4] G. Donoho, "Building a Web service to provide real-time stock quotes," MCAD.Net, pp.38–42, Feb. 2004.
[5] E. Durant, "Embedded real-time system considerations," EECS Department of Milwaukee School of Engineering, pp.1–19, April 1998.
[6] T. Xie, X. Qin, and A. Sung, "SAREC: A security-aware scheduling strategy for real-time applications on clusters," Proc. 34th International Conference on Parallel Processing, pp.5–12, Norway, June 2005.
[7] C. Wang and W.A. Wulf, "Towards a framework for security measurement," Proc. Twentieth National Information Systems Security Conference, pp.522–533, Baltimore, MD, Oct. 1997.
[8] P.A. Bernstein, "Middleware: A model for distributed system services," Commun. ACM, vol.39, no.2, pp.86–98, 1996.
[9] J. Wray, RFC2744- Generic Security Service API Version 2: C-bindings, http://www.faqs.org/rfcs/rfc2744.html, 2000.
[10] Programming Security for Java Applications, http://e-docs.bea.com/wles/docs42/programmersguide/
[11] C. Irvine and T. Levin, "Quality of security service," Proc. New Security Paradigms Workshop2000, pp.91–99, Cork, Ireland, Sept. 2000.
[12] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R.F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," 8th IEEE Heterogeneous Computing Workshop (HCW '99), pp.30–44, April 1999.
[13] A.S. Tanenbaum and M. Steen, Distributed Systems: Principles and Paradigms, ISBN 0130888931, Prentice Hall, 2002.
[14] T. Xie and X. Qin, "Enhancing security of real-time applications on grids through dynamic scheduling," Proc. 11th Workshop on Job Scheduling Strategies for Parallel Processing, pp.146–158, Cambridge, MA, USA, June 2005.
[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] E. Nahum, S. O'Malley, H. Orman, and R. Schroeppel, "Towards high performance cryptographic software," Proc. IEEE Workshop Architecture and Implementation of High Performance Communication Subsystems, pp.69–72, Aug. 1995.
[17] M. Harchol-Balter and A. Downey, "Exploiting process lifetime distributions for load balancing," ACM Trans. Comput. Syst., vol.3, no.31, pp.253–285, 1997.



Tao Xie is a Ph.D. Candidate in Computer Science at the New Mexico Institute of Mining and Technology in the USA. His research interests are security-aware scheduling, high performance computing, cluster and Grid computing, parallel and distributed systems, real-time/embedded systems, and information security.

Xiao Qin received the BS and MS degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively. He received the PhD degree in computer science from the University of Nebraska-Lincoln in 2004. Currently, he is an assistant professor in the department of computer science at the New Mexico Institute of Mining and Technology. His research interests are in parallel and distributed systems, storage systems, real-time computing, performance evaluation, and fault-tolerance. He is a member of the IEEE.