SUPPORT FOR SECURITY IN DISTRIBUTED SYSTEMS USING MESSIAHS

Steve J. Chapin
Department of Mathematics
and Computer Science
Kent State University
Kent, OH 44242-0001
sjc@cs.kent.edu

Eugene H. Spafford
COAST Laboratory
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–1398
spaf@cs.purdue.edu

Abstract

The MESSIAHS project is investigating the construction of a set of mechanisms to support task placement in autonomous, heterogeneous, distributed systems. In this paper we explore aspects of the MESSIAHS system that support security in distributed systems.

In particular, we will concentrate on aspects of MESSIAHS that defeat denial of service attacks, provide firewalls, protect private system description information, and support matching of tasks and systems based on security ratings. Development of these features will allow tasks to be scheduled in a heterogeneous distributed system, while protecting data and system integrity.

MESSIAHS is a set of mechanisms that ties together disparate computing resources to achieve distributed processing without sacrificing local control. MESSIAHS is novel in that it includes support for autonomous systems while providing flexible, scalable mechanisms to implement scheduling algorithms for heterogeneous distributed systems.

Keywords: distributed systems, scheduling, security, autonomy, availability, visibility

1 Introduction

We are investigating scheduling support mechanisms for autonomous, heterogeneous, distributed systems. Our goal is to develop mechanisms that allow scheduling algorithms to be implemented for large-scale distributed systems using heterogeneous hardware and software, across administrative boundaries. Such large-scale distributed systems can achieve performance surpassing that of the largest parallel supercomputers [11], and increase utilization of underutilized computing power [8]. As part of this work, we have developed a set of mechanisms and a prototype implementation called MESSIAHS: **M**echanisms **E**ffecting **S**cheduling **S**upport **I**n **A**utonomous, **H**eterogeneous **S**ystems [5, 4].

Our research is motivated by three factors. First, decentralization of computing systems has introduced administrative domains as a barrier to distributed computing. To overcome this, some method must be found to unite systems from incompatible administrative domains while respecting the autonomy of the individual systems. Second, many researchers have concentrated on scheduling and load-balancing algorithms while assuming the existence of the mechanisms necessary to support them (see, for example, Sarkar and Hennessy [13], Lo [12], or Blake [1]). They have either designed ad-hoc mechanisms to support particular algorithms, or limited their research to theoretical analysis of the scheduling algorithms. Third, users of computer systems may require resources that are not available locally, such as specialized processors or remote databases.

This paper concentrates on security aspects of the MESSIAHS mechanisms, which ties in with the first factor listed above. As part of the support for distributed computing across administrative domains, MESSIAHS provides mechanisms that

1. thwart denial of service attacks,
2. can act as a firewall to limit access by outside systems,
3. can restrict the flow of sensitive system description information outside an administrative domain,
4. and allows systems and tasks to be labeled in support of partitioning based on security requirements.

Section 2 gives background information describing the MESSIAHS system. Sections 3, 4, 5, and 6 describe the MESSIAHS mechanisms that support the four points listed above. Section 7 contains concluding remarks and proposes future directions for our investigation of security in distributed task placement.

2 MESSIAHS Background

Our systems are structured in a hierarchical fashion based on *virtual systems* representing administrative domains. A virtual system is composed of a set of subordinate virtual systems. Within each of these sets there can be many machines, which could be further grouped into virtual systems. At the lowest level, each machine is the sole member of a virtual system. We call an encapsulating virtual system a *parent*, and a subordinate system a *child*. Children with the same parent are called *siblings*.

For example, figure 1 displays part of the administrative structure of the Kent State University Mathematics and Computer Science Department. Within the department, there are several generally accessible machines such as Chaos and Nimitz, as well as machines supporting specialized research projects. One of these projects is

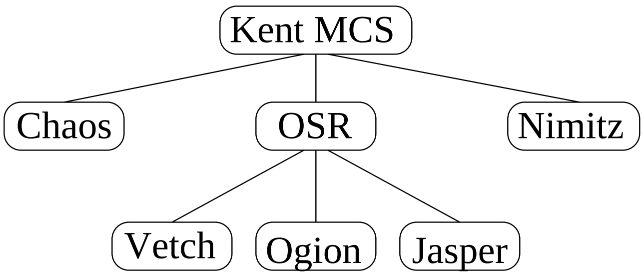

Figure 1: A subset of the machines in the Kent State Math/CS Dept.

the Operating Systems Research (OSR) project, which has administrative authority over a set of machines including Ogion, Vetch, and Jasper.

In MESSIAHS, each virtual system in the hierarchy has a scheduling support module that is responsible for maintaining the set of information required by the scheduling policy. Scheduling algorithms take a set of tasks and a description of the underlying multicomputer and devise an assignment of tasks to processors according to an optimizing criterion.

Our method for supporting scheduling decisions has three main parts: the *system description vector*, the *task description vector*, and the update protocol used to communicate between systems. The description vectors contain state description information, including system processing load, memory statistics, processing capabilities, and storage capacities. The update protocol sends system description vectors between modules.

The model for update flow is that a module collects several description vectors, adds information describing the local system, and condenses the resulting set of description vectors into one vector to facilitate scalability. This vector will be advertised to its parents and children. The module can also decide not to include data in the outgoing vector based on security constraints.

When a task is submitted for execution, a task description vector is sent to a scheduling module.[1] The scheduling module compares the task description to its own system description and the system description vectors it has received from other systems. Based on the scheduling policy, the module chooses one of the systems and attempts to schedule the task there.

MESSIAHS attempts to sacrifice the least autonomy for participating systems. There are four types of autonomy in distributed systems, as defined in [9, 6, 7], and refined in [4]: *execution autonomy*, *communication autonomy*, *design autonomy*, and *administrative autonomy*. Execution autonomy means that each system decides whether it will honor a request to execute a task; each system also has the right to revoke a task that it had previously accepted. Communication autonomy means that each system decides the content and frequency of state advertisements, and what other messages it sends. A system is not required to advertise all its capabilities, nor is it required to respond to messages from other systems. Design autonomy gives the architects of a system freedom to design and construct it without regard to existing

[1] These requests are called *scheduling requests*.

systems, yielding heterogeneous systems.

Administrative autonomy means that each system can have its own usage policies and behavioral characteristics, independent of any others. In particular, a local system can run in a manner counterproductive to a global optimum. In the usual case, scheduling modules will cooperate, but administrators must be free to set their local policies or they are unlikely to participate in the distributed system [2, 8].

The next four sections examine the behavior of the module and show how the autonomy support within MESSIAHS facilitates security in distributed systems.

3 Denial of Service Attacks

Communication, administrative, and execution autonomy form a basis to thwart denial of service attacks. Each system can autonomously decide whether or not to accept any task. Thus, policies can be written to use current load or the identity of a requesting system as criteria to screen incoming requests.

MESSIAHS implements two interface layers that scheduler-writers can use to implement their algorithms. The first, called the *MESSIAHS toolkit*, is a library of function calls that can be used with a high-level language such as C [3]. The second, the *MESSIAHS Interface Language*, or MIL, is an interpreted language that is especially tailored to the task of scheduling [5].

Either of these interface layers can be used to implement scheduling *filters*. A filter takes two description vectors and returns a numerical result indicating how well they match. A *task filter* compares an incoming task description vector to a system description vector and returns an integer. A negative number indicates an error during the evaluation of the filter, while zero indicates that there is no match. In either case, the task is not accepted for input. Positive integers indicate a match. In general, larger values imply a better match, although a boolean filter can be implementing by returning the same value for all matches, e.g. the integer one.

For example, the local policy could decline scheduling requests when the local load average exceeds a threshold. This would limit the impact of outside tasks on the system, although it would not discriminate between legitimate and malicious requests for resources. A policy based on the source of the request could ensure that the task comes from a trusted source. A mixture of these policies could limit the number of tasks from untrusted sources while also limiting the total load on the system. In this way, an attempted denial of service will consume at most a small percentage of the resources of the machine.

Communication autonomy can also help to defeat denial of service attacks. Because a system is not required to respond to a message, it can simply ignore suspicious scheduling requests. This diminishes the possibility of saturating the scheduling module with requests from outlaw systems. It also eliminates a possible covert channel, wherein an attacker could study the behavior of the system in response to spurious scheduling requests.

In addition, execution autonomy allows the scheduling policy to revoke or migrate running jobs. This facility can be used to remove tasks consuming excess resources, or to respond to a surge in load caused by an attempted denial of service attack.

4 Firewalls

It is sometimes desirable to mask the details of a resource, while still allowing outside access. This is commonly done for electronic mail systems, and is usually implemented through the use of a *firewall* [10]. All attempts to access a resource pass through the firewall, and the outside agent accessing the resource cannot tell the exact location of the resource.

For example, in figure 1, the OSR node can act as a firewall to hide the presence of Vetch, Ogion, and Jasper. It can still advertise some of their capabilities to the other nodes in the system, but it appears as if all their resources are located at the OSR node.

MESSIAHS incorporates two mechanisms to accomplish this: information condensation and proxy acceptance. Information condensation takes place when two or more update vectors are combined to form a single vector for advertisement. For example, OSR combines the capabilities of OSR, Vetch, Ogion, and Jasper into a single vector that can be sent to nodes outside the virtual system rooted at OSR. In the process, all identifying information, such as location information of individual resources, is removed.

For example, suppose Ogion were an SGI Indy running IRIX 5.1 [2], Vetch were a SPARC IPC running SunOS 4.1[3], and Jasper were a 486 clone running FreeBSD. The information advertised by OSR would indicate the presence of MIPS, 486, and SPARC processors, as well as the presence of the IRIX, BSD, SunOS operating systems. There is no indication which processor is running which operating system. The OSR node knows this, but does not advertise it to the outside world. This might cause another node to send a spurious request to OSR, e.g. a request to run a task on a SPARC processor running the BSD operating system. However, OSR will have enough information to discard the request, and no tasks will be misscheduled as a result.

This leaves open the question, "If a task is scheduled on a system, how is it moved to the system without the originator knowing where the system is?" The solution used in MESSIAHS is the *proxy accept*. When passing a scheduling request to an interior node, the firewall logs the request, replaces the originator's address with its own address, and waits for the response from the interior node. If the node accepts the request, it sends an acceptance message back to the firewall.

Upon receipt of the accept message, the firewall replaces the address of the acceptor with its address, and forwards the acceptance to the originator of the request. The originator then treats the firewall as the acceptor, and forwards the task for execution. The firewall then forwards the task to the real acceptor, and continues to act as an intermediary between the acceptor and the outside world.

[2] Indy and IRIX are trademarks of Silicon Graphics, Incorporated.

[3] SPARC and SunOS are trademarks of Sun Microsystems, Inc.

```
struct statvec {
    float min, max, mean, stddev, total;
};

typedef struct statvec Statvec;

struct procclass {
  bit32     nsys;      /* number of machines in this class */
  Statvec   qlen;      /* run queue statistics */
  Statvec   busy;      /* load on cpu (percentage) */
  Statvec   physmem;   /* total physical memory */
  Statvec   freemem;   /* available memory */
  Statvec   specint92; /* ratings for specint 92 */
  Statvec   specfp92;  /* ratings for specfp 92 */
  Statvec   freedisk;  /* public disk space statistics */
};
```

Figure 2: Statistics vectors and processor classes in MESSIAHS

5 Control of Advertised Information

To be secure, systems must not advertise sensitive information to untrusted systems. The communication autonomy support in MESSIAHS allows scheduling policies to omit data from their outgoing vectors. This feature can be used to filter outgoing data to be consistent with a security policy.

To facilitate scalability, MESSIAHS uses a statistical representation of the capabilities of a virtual system. That is, instead of listing specific ratings of individual machines, the minimum, maximum, mean and standard deviation for a capability are kept, as well as the number of systems represented in a vector (see figure 2).

To partition the possible space of attributes, machines are divided into classes based on logarithmic scale of their processor speed, with a structure containing statistical information regarding the available resources for machines in each class (see figure 2). In this way, information can be condensed while still providing enough information for scheduling algorithms to make intelligent choices.

MESSIAHS provides routines to automatically combine multiple statistical vectors into one. This is the mechanism used by the module to coalesce multiple system descriptions into the description of a single virtual system. The autonomy support within the mechanisms allows fields to be omitted from the combination. For example, if the OSR project administrator does not want the capabilities of Jasper advertised to nodes outside the project, he can specify that Jasper's resources not be included in OSR's advertised vector. Both MIL and the scheduling toolkit allow the administrator to restrict information advertisement in this fashion.

```
begin combining
   string $out.tier        not match($out.tier, "preferred"):
                           set $out.tier + ":preferred";
   string $out.department  not match($out.department, "research"):
                           set $out.department + ":research";
end
```

Figure 3: A code fragment from MIL using labels

The obvious tradeoff is this scheme is the size of the advertised vectors versus the degree of detail present in the vectors. The approach taken allows the vectors to be kept to a reasonable size[4] while still providing sufficient visibility of individual machines so that scheduling algorithms can function well.

6 Extension and Labeling Support

In addition to the fixed data represented by statistical vectors, MESSIAHS also allows administrators to extend the system description vector. This affords the mechanisms flexibility in supporting scheduling algorithms, and can be used to support secure processing based on security classifications.

Systems can insert labels in their extension vectors to indicate the security classification required to run a task on that system. Tasks can include a security label listing their security classification. The scheduling algorithm can match the levels to ensure that the task's security rating is equal to or higher than that of the system.

The MESSIAHS mechanisms can improve the efficiency of a distributed computation. Large jobs can be partitioned into smaller tasks based on their security requirements, and then only those tasks that require secure processing will be run on secure sites. Tasks that do not require secure processing can be run on any general-purpose processor within the distributed system. This not only reduces the load on the secure installations, it increases the security of these systems by ensuring that only computations that require secure resources are run there.

This labeling mechanism could also be used in commercial systems. Within a single organization, tasks could be labeled with their department of origin, e.g. sales or research. Systems could protect private data by only executing certain classes of jobs. This mechanism could also be used by an institution that sells processing time to outside customers. The institution could offer different tiers of service, and jobs from customers would be labeled based on the tier they had purchased. Jobs from more expensive tiers might receive preferential treatment by being given higher priority, or being assigned to faster computers.

Figure 3 shows an example usage of labeling written in MIL. Assume that the

[4] The update vectors in the prototype implementation are approximately two kilobytes in size.

intent is to advertise that the virtual system will run tasks for preferred customers within the research department. This code fragment makes sure that the service tier preferred appears in the outgoing description vector, and ensures that the research department label also appears. Again, outside systems cannot determine if the preferred tier applies to the research department, but this will not cause a breach of security.

Two factors complicate the use of MESSIAHS for this type of service. First, there must be some method of ensuring that machines and tasks cannot spoof higher security classifications or service tiers. Second, there must be guarantees that the data in the extension area remains private and uncorrupted, because communication autonomy allows intervening systems to read or alter the contents of an advertisement. In the absence of a distributed secure network, we are left to devise software solutions to these problems.

We can use well-known authentication techniques such as those found in Kerboros [14] to ensure the validity of labels. A possible solution to the second problem is to encrypt private data within the extended portion of the task description vector so that only trusted hosts can view the secret data. However, this scheme presents the difficulty that intermediate nodes have no semantic knowledge of the encrypted information, and therefore cannot apply any combining rules to condense the information. Finding a clean solution to this dilemma is an open problem.

7 Concluding Remarks

We have described the MESSIAHS system for scheduling support. MESSIAHS includes generous support for autonomy in distributed systems, and this autonomy support can form the basis for security measures.

We have shown how the scheduling support mechanisms can support four aspects of security: thwarting denial of service attacks, acting as a firewall, restricting the flow of information outside an administrative domain, and allowing systems and tasks to be matched based on their security requirements.

The MESSIAHS system has several potential applications for distributed systems in a trusted environment. The mechanisms can support process migration and load balancing. Because the update protocols track which machines are available, fault tolerance can be layered over the mechanisms. The revocation facility can support transaction management in a nested-transaction environment.

Our plans for the future are to study the issue of cryptographic techniques to handle end-to-end security issues. However, there are significant barriers to be overcome to prevent nodes from advertising encrypted, sensitive information outside an administrative domain.

References

[1] B. A. Blake. Assignment of Independent Tasks to Minimize Completion Time. *Software–Practice and Experience*, 22(9):723–734, September 1992.

[2] A. Bricker, M. Litzkow, and M. Livny. Condor Technical Summary. Technical Report 1069, Department of Computer Science, University of Wisconsin-Madison, January 1992.

[3] S. Chapin and E. Spafford. Implementing Scheduling Algorithms Using MESSIAHS. *Scientific Programming*, 1994. to appear in a special issue on Operating System Support for Massively Parallel Computer Architectures.

[4] S. J. Chapin. Scheduling Support Mechanisms for Autonomous, Heterogeneous, Distributed Systems. Ph.D. Dissertation, Purdue University, 1993.

[5] S. J. Chapin and E. H. Spafford. Constructing Distributed Schedulers with the MESSIAHS Interface Language. In *27th Hawaii International Conference on Systems Sciences*, volume 2, pages 425–434, Maui, Hawaii, January 1994.

[6] W. Du, A. K. Elmagarmid, Y. Leu, and S. D. Ostermann. Effects of Local Autonomy on Global Concurrency Control in Heterogeneous Distributed Database Systems. In *Second International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, pages 113–120. IEEE, 1989.

[7] F. Eliassen and J. Veijalainen. Language Support for Multidatabase Transactions in a Cooperative, Autonomous Environment. In *TENCON '87*, pages 277–281, Seoul, 1987. IEEE Regional Conference.

[8] C. A. Gantz, R. D. Silverman, and S. J. Stuart. A Distributed Batching System for Parallel Processing. *Software–Practice and Experience*, 19, 1989.

[9] H. Garcia-Molina and B. Kogan. Node Autonomy in Distributed Systems. In *ACM International Symposium on Databases in Parallel and Distributed Systems*, pages 158–166, Austin, TX, December 1988.

[10] S. Garfinkel and E. Spafford. *Practical UNIX Security*. O'Reilly and Associates, 1991. ISBN 0-937175-72-2.

[11] A. H. Karp, K. Miura, and H. Simon. 1992 Gordon Bell Prize Winners. *IEEE Computer*, 26(1):77–82, January 1993.

[12] V. M. Lo. Task Assignment to Minimize Completion Time. In *Distributed Computing Systems*, pages 329–336. IEEE, 1985.

[13] V. Sarkar and J. Hennessy. Partitioning Parallel Programs for Macro-Dataflow. In *ACM Conference on Lisp and Functional Programming*, pages 202–211, August 1986.

[14] A J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Usenix Conference Proceedings*, pages 191–202, Dallas, TX, February 1988.