Integrating Trust into Grid Resource Management Systems

Farag Azzedin and Muthucumaru Maheswaran

University of Manitoba and TR*Labs*
Winnipeg, Manitoba
Canada
E-mail: {fazzedin, maheswar}@cs.umanitoba.ca

Abstract

Grid computing systems that have been the focus of much research activities in recent years provide a virtual framework for controlled sharing of resources across institutional boundaries. Security is one major concern in any system that enables remote execution. Several techniques can be used for providing security in Grid systems including sandboxing, encryption, and other access control and authentication mechanisms. The additional overhead caused by these mechanisms may negate the performance advantages gained by Grid computing. Hence, we contend that it is essential for the scheduler to consider the security implications while performing resource allocations. In this paper, we present a trust model for Grid systems and show how the model can be used to incorporate the security implications into scheduling algorithms. Three scheduling heuristics that can be used in a Grid system are modified to incorporate the trust notion and simulations are performed to evaluate the performance.

Keywords: Grid computing, resource management system, security, trust.

1. Introduction

Resource management in Grid systems [5, 9] is challenging due to: (a) geographical distribution of resources, (b) resource heterogeneity, (c) autonomously administered Grid domains having their own resource policies and practices, and (d) Grid domains using different access and cost models.

In previous generation *distributed computing environments* (DCEs), *resource management systems* (RMSs) were primarily responsible for allocating resources for tasks. They also performed functions such as resource discovery and monitoring that supported their primary role. In Grid systems, with distributed ownership for the resources and tasks, it is important to consider *quality of service* (QoS) and security while allocating resources. Integration of QoS into RMS has been examined by several researchers [7, 11]. However, security is implemented as a separate subsystem of the Grid [6] and the RMS makes the allocation decisions oblivious of the security implications.

Our study on integrating trust into resource management algorithms is motivated by the following scenarios. Grid computing systems provide a facility that enable large-scale controlled sharing and interoperation among resources that are distributively owned and managed. Trust is a major concern of the consumers and producers of services that participate on a Grid. Some resource consumers may not want their applications mapped onto resources that are owned and/or managed by entities they do not trust. Similar concerns apply from the resource producer side as well. Current generation of distributed systems addresses these concerns by providing security at different levels. Suppose resource M is allocated task T. Resource M can employ sandboxing techniques to prevent task T from eavesdropping or interfering with other computation or activities ongoing on M. Similarly, task T may employ encryption, data hiding, intelligent data encoding, or other mechanisms to prevent M from snooping into the sensitive information carried by task T.

Based on the above scenarios we hypothesize that if the RMS is aware of the security requirements of the resources and tasks it can perform the allocations such that the security overhead is minimized. This is the goal of the *trust-aware resource management system* (TRMS) studied here. The TRMS achieves this goal by allocating resources considering a trust relationship between the *resource provider* (RP) and the *resource consumer* (RC). If an RMS maps a resource request strictly according to the trust, then there can be a severe load imbalance in a large-scale wide area system such as the Grid. On the other hand, considering just the load balance or resource-task affinities, as in existing RMSs, causes inefficient overall operation due to the introduction of the overhead caused by enforcing the required level of security. Mapping according to load balance or trust considerations results in diverging schedules. The former spreads the requests for the sake of load balance while the latter segregates them for security considerations. In the TRMS algorithms examined here, the minimization criterion is derived from load balancing and security considerations.

This paper is organized as follows. Section 2, defines the notions of trust and reputation and outlines mechanisms for computing them. A trust model for Grid systems is presented in Section 3. Trust-aware resource management algorithms are presented in Section 4. The performance and

the analysis of the proposed algorithms are examined in Section 5. Related work is briefly discussed in Section 6.

2. Trust and Reputation

2.1. Definition of Trust and Reputation

The notion of trust is a complex subject relating to a *firm belief* in attributes such as reliability, honesty, and competence of the trusted entity. There is a lack of consensus in the literature on the definition of trust and on what constitutes trust management [12, 8, 1]. The definition of trust that we will use in this paper is as follows:

> *Trust is the firm belief in the competence of an entity to act as expected such that this firm belief is not a fixed value associated with the entity but rather it is subject to the entity's behavior and applies only within a specific context at a given time.*

That is, the *firm belief* is a dynamic value and spans over a set of values ranging from *very trustworthy* to *very untrustworthy*. The *trust level* (TL) is built on past experiences and is given for a specific context. For example, entity y might trust entity x to use its storage resources but not to execute programs using these resources. The TL is specified for a given time frame because the TL today between two entities is not necessarily the same TL a year ago.

When making trust-based decisions, entities can rely on others for information pertaining to a specific entity. For example, if entity x wants to make a decision of whether to have a transaction with entity y, which is unknown to x, x can rely on the reputation of y. The definition of reputation that we will use in this paper is as follows:

> *The reputation of an entity is an expectation of its behavior based on other entities' observations or the collective information about the entity's past behavior within a specific context at a given time.*

2.2. Computing Trust and Reputation

In computing trust and reputation, several issues have to be considered. First, the trust decays with time. For example, if x trusts y at level p based on past experience five years ago, the trust level today is very likely to be lower unless they have interacted since then. Similar time-based decay also applies for reputation. Second, entities may form alliances and as a result would tend to trust their allies more than they would trust others. Finally, the TL that x holds about y is based on x's direct relationship with y as well as the reputation of y, i.e., the trust model should compute the eventual trust based on a combination of direct trust and reputation and should be able to weigh the two components differently.

Let x and y denote two entities. The trust relationship for a specific context c at a given time t between the two entities, expressed as $\Gamma(x, y, t, c)$, is computed based on the direct relationship for the context c at time t between x and y, expressed as $\Theta(x, y, t, c)$, as well as the reputation of y for context c at time t expressed as $\Omega(y, t, c)$. Let the weights given to direct and reputation relationships be α and β, respectively. The trust relationship is a function of direct trust and reputation. If the "trustworthiness" of y, as far as x is concerned, is based more on direct relationship with x than the reputation of y, α will be larger than β.

The direct relationship is computed as a product of the TL in the *direct-trust table* (DTT) and the *decay function* ($\Upsilon(t - t_{xy}, c)$), where c is the context, t the current time, and t_{xy} the time of the last transaction between x and y. The reputation of y is computed as the average of the product of the TL in the *reputation-trust table* (RTT), the *decay function* ($\Upsilon(t - t_{zy}, c)$), and the recommender trust factor ($R(z, y)$) for all entities $z \neq x$. In practical systems, entities will use the same information to evaluate direct relationships and give recommendations, i.e., RTT and DTT will refer to the same table. Because reputation is based primarily on what other entities say about a particular entity, we introduced the *recommender trust factor* R to prevent cheating via collusions among a group of entities. Hence, R is a value between 0 and 1 and will have a higher value if the recommender does not have an alliance with the target entity. In addition, we assume that R is an internal knowledge that each entity has and is learned based on actual outcomes.

$$\begin{aligned} \Gamma(x, y, t, c) &= \alpha \times \Theta(x, y, t, c) + \beta \times \Omega(y, t, c) \\ \Theta(x, y, t, c) &= DTT(x, y, c) \times \Upsilon(t - t_{xy}, c) \end{aligned}$$

And $\forall\, z \neq x$, we have:

$$\Omega(y, t, c) = \frac{\sum_{k=1}^{n} RTT(z, y, c) \times R(z, y) \times \Upsilon(t - t_{zy}, c)}{\sum_{k=1}^{n}(z)}$$

Currently, we are developing a trust management architecture that can evolve and maintain the trust values based on the concepts explained above. The rest of this paper is concerned with using the trust values maintained by such a system to perform efficient resource allocation.

3. A Trust Model for Grid Systems

3.1. Trust Model for Grid Systems

In our model, the overall Grid system is divided into *Grid domains* (GDs). The GDs are autonomous administrative entities consisting of a set of resources and clients managed by a single administrative authority. By organizing a Grid as a collection of GDs, issues such as scalability,

site autonomy, and heterogeneity can be easily addressed. In our model, we associate two virtual domains with each GD: (a) a *resource domain* (RD) to signify the resources within the GD and (b) a *client domain* (CD) to signify the clients within the GD. As RDs and CDs are virtual domains mapped onto GDs, some instances of RDs and CDs can map onto the same GD.

An RD has the following attributes that are relevant to the TRMS: (a) ownership, (b) set of *type of activity* (ToA) it supports, and (c) *trust level* (TL) for each ToA. The set of ToAs determine the functionalities provided by the resources that are part of the RD. Some example activities a task can engage at an RD include printing, storing data, and using display services. Associating a TL with each ToA provides the flexibility to selectively open services to clients.

Similarly, the CDs have their own trust attributes relevant to the TRMS. The CD trust attributes include: (a) ownership, (b) ToAs sought, and (c) TLs associated with ToAs. The ToA field indicates the type and number of activities a client is requesting. The ToAs can be atomic or composed. A client with an atomic ToA requires just one activity whereas a client with a composed ToA requires multiple activities.

A trust level table exists between a set of RDs and CDs. The entries in the trust level table are *symmetric* quantifiers for the trust relationships that are asymmetric. For example, let the trust relationship between client domain CD_i and resource domain RD_j be defined by $f(i,j)$. Because trust is an asymmetric function the reverse relationship between RD_j and CD_i, in general, is not given by $f(i,j)$. However, in this study, we denote the current value of the two functions using a single value, i.e., TL_{ij}^{k} for CD_i and RD_j engaging in activity A_k. The entry TL_{ij}^{k} in a trust level table denotes the trust value for an activity of a client from CD_i on a resource in RD_j. Suppose we have client X from CD_i wanting to engage in activities A_p, A_q, and A_r on resource Y at RD_j. From the trust level table, we can compute the *offered trust level* (OTL), TL_{ij}^{o} for the composite activity between X and Y, i.e., $\mathrm{TL}_{ij}^{o} = \min(\mathrm{TL\ for\ A}_p, \mathrm{TL\ for\ A}_q, \mathrm{TL\ for\ A}_r)$. There are two *required trust levels* (RTLs), one from the client side and the other from the resource side. If the OTL is greater than or equal to the maximum of client and resource RTLs, then the activity can proceed with no additional overhead. Otherwise, there will be additional security overhead involved in supplementing the OTL to meet the requirements.

The trust level values used in Table 1 range from *very low trust level* denoted as *A*, to *extremely high trust level* denoted as *F*. Table 1 shows the *expected trust supplement* (ETS) for different RTL and OTL values. The ETS values are given by $\mathrm{RTL} - \mathrm{OTL}$. The ETS value is zero, when $\mathrm{RTL} - \mathrm{OTL} < 0$. It can be noted from Table 1 that the RTL has a value F that is not provided by OTL. This is supported in the model so that client or resource domains can enforce enhanced security by increasing their RTL value to F.

Table 1. Expected trust supplement values.

requested TL	offered TL				
	A	B	C	D	E
A	0	0	0	0	0
B	B - A	0	0	0	0
C	C - A	C - B	0	0	0
D	D - A	D - B	D - C	0	0
E	E - A	E - B	E - C	E - D	0
F	F	F	F	F	F

A straight forward approach to creating and maintaining the trust level table can result in an inefficient process in a very large-scale system such as the Grid. This process is made efficient in our model by various methods. First, as mentioned previously, we divide the Grid system into GDs. The resources and clients within a GD inherit the parameters associated with the RD and CD that are associated with the GD. This increases the scalability of the overall approach. Second, trust is a slow varying attribute, therefore, the update overhead associated with the trust level table is not significant. A value in the trust level table is modified by a new trust level value that is computed based on a *significant* amount of transactional data.

Figure 1 shows a block diagram of a trust-aware RMS. The CDs and RDs have agents associated with them that monitor the Grid level transactions and form the trust notions. These agents have access to the trust level table. If the new trust values they form are different from the existing values in the tables, the agents update the table. In this study, we maintain a single table in a centrally organized RMS. The table may, however, be replicated at different domains for reading purposes.

4. Trust-Aware Resource Management Algorithms

4.1. Overview

In this section, we present three *trust-aware resource management* (TRM) algorithms as example applications of integrating trust into the RMS where clients belonging to different CDs present the requests for task executions and the TRM algorithms allocate the resources. Different requests belonging to the same CD may be mapped onto different RDs. The TRM algorithms presented here are based on the following assumptions: (a) scheduler is organized centrally, (b) tasks are mapped non-preemptively, and (c) tasks are indivisible (i.e., a task cannot be distributed over multiple machines).

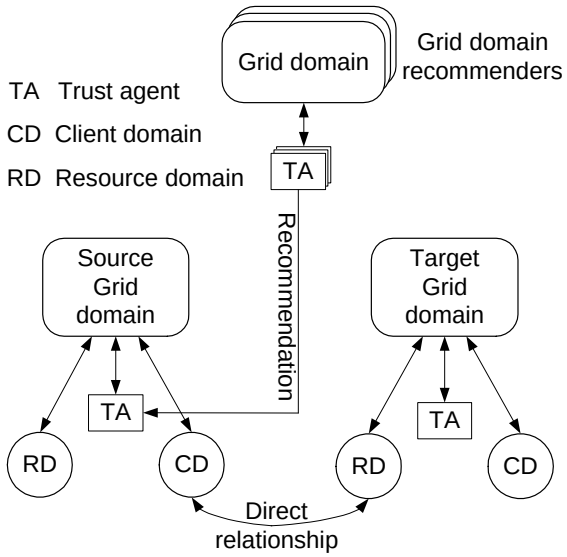

Figure 1. Components of a Grid resource management trust model.

Our three TRM algorithms are implemented using three heuristics based on [10]: (a) trust-aware *minimum completion time* (MCT) heuristic, (b) trust-aware Min-min heuristic, and (c) trust-aware Sufferage heuristic. The MCT is an on-line or immediate mode mapping heuristic whereas the Min-min and Sufferage are batch mode mapping heuristics.

For the on-line mode mapping heuristic, the TRM schedules client requests as they arrive. This scheduling is done by the `TRM-scheduler` algorithm based on the MCT on-line mapping heuristic. For the batch mode mapping heuristics, the TRM collects client requests for a predefined time interval to form batch of requests, called a `meta-request`. The `TRM-scheduler` algorithm schedules the `meta-request` based on the two batch heuristics namely trust-aware Min-min, and trust-aware Sufferage heuristics.

Let $t(r_i)$ denote the task being executed by request r_i and $c(r_i)$ denote the originating client. Furthermore, let R_i be the i^{th} meta-request and α_i be the available time of machine M_i after executing all requests assigned to it. Further, α_i^j be the available time α_i after executing all requests that belong to meta-request R_j. Also, let EEC($M_i, t(r_j)$) be the *expected execution cost* for $t(r_j)$ on machine M_i and ESC($M_i, t(r_j)$) be the *expected security cost* if $t(r_j)$ is assigned to machine M_i. The ESC value is a function of the *trust cost* (TC) value obtained from ETS (Table 1) and the task under consideration. When the RMS is considering the trust notion while allocating resources, the following equation is used to calculate the ESC table:

$$ESC(M_i, t(r_j) \quad = \quad EEC(M_i, t(r_j) \times (TC \times 15)/100$$

If the RMS is not considering the trust notion while allocating resources, the following equation is used to calculate the ESC table:

$$ESC(M_i, t(r_j) \quad = \quad EEC(M_i, t(r_j) \times 50/100$$

The trust levels A to F are assigned corresponding numeric values that range from 1 to 6, respectively. As shown in Table 1, TC ranges from 0 to 6. Hence the average TC value is 3. In our model, the ESC values are computed by multiplying the EEC values by a weighted TC value. We arbitrarily choose the weight for TC as 15. Therefore, when trust is considered, on average the ESC values are calculated as 45% of the EEC. On the other hand, when trust is not considered the ESC values are calculated as 50% of the EEC.

Finally, let ECC($M_i, t(r_j)$) denotes the *expected completion cost* of $t(r_j)$ on machine M_i which is computed as the EEC of $t(r_j)$ on machine M_i plus the ESC of $t(r_j)$ on machine M_i. The goal of TRM algorithm is to assign R_i = $\{r_0 \ldots r_{n-1}\}$ such that $\{max_m\{\alpha_m^i\}\}$ is minimized $\forall_m$ where n is the number of requests and m is the number of machines.

The Trust-Aware Minimum Completion Time Algorithm: The MCT heuristic [10] assigns each task to the machine that results in that task's earliest completion time. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. As a task arrives, all the machines are examined to determine the machine that gives the earliest completion time for the task.

The trust-aware MCT algorithm starts by computing the ESC in terms of the trust cost which is the difference between the $c(r_j)$ requested TL and the offered TL by a machine M_i in RD_k. The trust cost is an indicator of how well is the trust relationship between an RD and a CD. For example, if the trust cost is 0, then the two parties completely trust each other. After that, the ECC table is initialized and the request r_j is assigned to the machine with the lowest completion cost. The task $t(r_j)$ that was successfully assigned to machine M_i is used to update machine M_i available time α_i which in turn is used to compute or update the expected completion cost for all requests yet to be assigned to machime M_i.

The Trust-Aware Min-min Algorithm: The `TRM-scheduler` algorithm schedules a batch of requests called *meta-request*. To map the meta-requests, we introduce a heuristic based on [10] called the trust-aware Min-min heuristic. Min-min begins by scheduling the tasks that change the expected machine available time by the least amount.

The initialization phase of he trust-aware Min-min algorithm is similar to the ones in the MCT heuristic. The request scheduled on machine M_i is deleted from the meta-request R_v. The task $t(r_j)$ that was successfully assigned

Table 2. Secure versus regular transmission for a 100 Mbps network.

File size/MB	Using rcp/(sec)	Using scp/(sec)	Overhead
1	0.19	0.63	69.84%
10	1.37	2.45	44.08%
100	9.77	15.34	36.31%
500	48.88	77.56	36.70%
1000	97.00	155.07	37.45%

Table 3. Secure versus regular transmission for a 1000 Mbps network.

File size/MB	Using rcp/(sec)	Using scp/(sec)	Overhead
1	0.34	0.65	47.69%
10	0.50	2.18	77.06%
100	4.98	14.23	65.00%
500	22.44	69.86	67.88%
1000	46.05	138.30	66.70%

to machine M_i is used to update machine M_i available time α_i which in turn is used to compute or update the expected completion cost for all requests yet to be assigned to machime M_i.

The Trust-Aware Sufferage Algorithm: The `TRM-scheduler` algorithm schedules a batch of requests called meta-request based on [10] called the trust aware Sufferage heuristic. The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it.

The initialization of the trust-aware Sufferage algorithm is similar to the Min-min heuristic. However, for each iteration, the algorithm picks an arbitrary request r_i from the meta-request and aasigns it to a machine m_j that gives the earliest completion cost for request r_i. If however there was another request r_k that was assigned to machine m_j previously, the algoeithm chooses the request (among r_i and r_k) that suffers the most if not assigned to machine m_j. It shoud be noted that the unchosen request (among r_i and r_k) will not be considered again for execution until the next iteration.

5. Performance Evaluation

5.1. Evaluation of Security Overheads

We conducted a study to examine the overhead of securing data transmissions for 100 Mbps and 1000 Mbps networks. The machines used were base on an Intel Pentium III processor running at 866 MHz with memory size of 256 MB and a level 2 cache of size 256 KB. Tables 2 and 3 show the security overhead for secure transmissions using *secure copy* (scp) versus the regular transmission using *remote copy* (rcp) for different network speeds and with different file sizes. As illustrated in Tables 2 through 3, using `scp` introduces an overhead caused by the addition of security to the file transfer.

From Table 3, we observe that the security overhead negates the benefits of using the high speed network. Also, the security overhead as shown in Table 3 is significant for the secure transmission when compared to the regular transmission using `rcp`.

Furthermore, a performance study was done in [4] where three target benchmark applications are processed by *Minimal i386 Software Fault Isolation Tool* (MiSFIT) [13] and *Security Automata SFI Implementation* (SASI x86SFI) [4] sandboxing systems. *Software fault isolation* (SFI) is a sandboxing technique for transforming code written in unsafe language into safe compiled code. `MiSFIT` specializes the `SFI` technique to transform C++ code into safe binary code whereas `SASI x86SFI` specializes `SFI` to transform x86 assembly language output of the GNU gcc C compiler to safe binary code. The three target applications used are: (a) a memory intensive application benchmark called *page-eviction hotlist*, (b) *logical log-structured disk*, and (c) a command line message digest utility called `MD5`.

Page-eviction hotlist has the highest runtime overhead of 137% on `MiSFIT` and 264% on `SASI x86SFI` compared to the execution of the target applications on the target systems with no sandboxing. The other two benchmark applications performed as follows (compared to their execution on the target systems with no sandboxing): the *logical log-structured disk* has runtime overhead of 58% on `MiSFIT` and 65% on `SASI x86SFI`, whereas `MD5` has runtime overhead of 33% on `MiSFIT` and 36% on `SASI x86SFI`.

The additional overhead caused by techniques such as sandboxing may negate the performance advantages gained by the Grid computing and hence we contend that it is essential for the scheduler to consider the security implications while performing resource allocations.

5.2. Analysis of the Trust-Aware Schemes

The goal of the three mapping heuristics (MCT, Min-min, and Sufferage) is to minimize the makespan, where makespan is defined as the maximum among the available times of all machines after they complete the tasks assigned to them. Initially $\alpha_m = 0, \forall_m$. The scheduler assigns request r_n to machine M_m such that the scheduling criterion is minimized. The heuristics considered in this paper use makespan minimization as their scheduling criterion.

Let X_{km} be the mapping function computed by the scheduler, where $X_{km} = 1$, if request r_k is assigned to machine M_m and 0, otherwise. The makespan $\Lambda = max_{M_m}\{\alpha_m\}$, where α_m is the available time of machine M_m after completing all the tasks assigned to it by the scheduler. The value of α_m is given by:

$$\begin{aligned}\alpha_m &= \sum_{k=0}^{n-1} ECC(t(r_k), m) \times X_{km} \\ &= \sum_{k=0}^{n-1} [EEC(t(r_k), m) + ESC(t(r_k), m)] \times X_{km}\end{aligned}$$

A given scheduling heuristic computes a value of X_{km} such that the makespan is minimized. It should be noted that due to the non-optimality of the heuristics, the makespan value may not be the globally minimal one.

Theorem: The makespan obtained a trust-aware scheduler is always less than or equal to the makespan obtained by the trust-unaware scheduler that uses the same assignment heuristic.

Proof: Let the makespan obtained by the trust-aware heuristic be:

$$\Lambda_T^n = max\{\sum_{k=0}^{n-1}(EEC(t(r_k), m) + ESC(t(r_k), m)) \times X_{km}^T\}$$

Let the makespan obtained by the trust-unaware heuristic be:

$$\Lambda_{UT}^n = max\{\sum_{k=0}^{n-1}(EEC(t(r_k), m) + ESC(t(r_k), m)) \times X_{km}^{UT}\}$$

For $n = 1$, i.e., for the first task,

$$\begin{aligned}\Lambda_T^1 &= (EEC(t(r_k), m) + ESC(t(r_k), m)) \times X_{km}^T \\ \Lambda_{UT}^1 &= (EEC(t(r_k), m) + ESC(t(r_k), m)) \times X_{km}^{UT}\end{aligned}$$

Suppose, $\Lambda_T^1 > \Lambda_{UT}^1$, and thus we will have the following inequality:

$$(EEC(t(r_k), m) + ESC(t(r_k), m)) \times X_{km}^T > (EEC(t(r_k), m) + ESC(t(r_k), m)) \times X_{km}^{UT}$$

X_{km}^T was chosen to minimize $(EEC(t(r_k), m) + ESC(t(r_k), m)) \times X_{km}^T)$ while X_{km}^{UT} was chosen to minimize $(EEC(t(r_k), m) + ESC(t(r_k), m)) \times X_{km}^{UT})$. The above inequality, implies another choice that further minimizes the sum exists that was not selected by the heuristic. This is a contradiction. Hence, $\Lambda_T^1 < \Lambda_{UT}^1$.

Let $\Lambda_T^i < \Lambda_{UT}^i$ (i.e., assume the trust-aware scheme provides a smaller makespan after mapping i tasks). Following the above process, we can show that $\Lambda_T^{i+1} < \Lambda_{UT}^{i+1}$. Therefore, by induction $\Lambda_T^n < \Lambda_{UT}^n$.

5.3. Evaluation of the Trust-Aware Schemes

Simulations were performed to investigate the performance of the trust aware resource management algorithms. The resource allocation process was simulated using a discrete event simulator with the requests arrivals modeled using a Poisson random process. The number of CDs and RDs were randomly generated from [1, 4]. The ToAs required for each request were randomly generated from [1, 4] meaning that each $t(r_i)$ involves at least one ToA but no more than four ToAs. The two RTL values were randomly generated from [1, 6] representing trust levels A to F, respectively. Whereas, the OTL values were randomly generated from [1, 5] representing trust levels A to E, respectively.

In an ECC matrix, the numbers along a row indicate the estimated expected completion cost of the corresponding request on different machines. The average variation along the rows is referred as the *machine heterogeneity*. Similarly, the numbers along a column of the ECC matrix indicate the estimated expected completion cost of the machine for different requests. The average variation along columns is referred to as *task heterogeneity*. Two classes of EEC matrices were used in the simulation. The first class is the consistent *low task and low machine heterogeneity* (LoLo). This class of ECCs model network computing systems that have related machines that are similar in performance. The tasks that are submitted to the system have similar resource requirements as well. The second class is the inconsistent LoLo. In this class, the machines are not related.

Table 4. Comparison of average completion time for inconsistent LoLo heterogeneity using the MCT heuristic.

# of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement
50	No	92.86%	5,817.38	36.99%
	Yes	93.56%	3,665.23	
100	No	96.29%	11,244.77	37.59%
	Yes	96.12%	7,018.38	

When not using trust, the idea is to map a task belonging to request r_i to machine M_j that gives us the earliest completion time without considering the security overhead. Although the completion time was calculated in terms of the execution time of $t(r_i)$ on M_j plus the security overhead of executing $t(r_i)$ on M_j, the security overhead is not considered when mapping $t(r_i)$ to M_j. For the trust-aware heuristics, the security overhead is considered when mapping as well as when calculating the completion time of executing $t(r_i)$ on M_j.

Tables 4 and 5, show the benefit of integrating the trust

Table 5. Comparison of average completion time for consistent LoLo heterogeneity using the MCT heuristic.

# of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement
50	No	93.90%	4,786.27	34.44%
	Yes	93.96%	3,137.78	
100	No	96.51%	9,117.53	34.26%
	Yes	96.81%	5,994.25	

notion into an MCT-based RMS. Table 4 was run for the inconsistent LoLo heterogeneity with 5 machines. In Table 4, the completion time was reduced by about 38%. Table 5 was run for the consistent LoLo heterogeneity with 5 machines. In Table 5, the completion time was reduced by about 35%.

Table 6. Comparison of average completion time for inconsistent LoLo heterogeneity using the Minmin heuristic.

# of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement
50	No	90.56%	3,983.04	23.51%
	Yes	90.87%	3,046.79	
100	No	93.71%	7,227.78	23.34%
	Yes	94.35%	5,540.47	

Table 7. Comparison of average completion time for consistent LoLo heterogeneity using the Minmin heuristic.

# of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement
50	No	93.17%	3,750.59	25.28%
	Yes	92.53%	2,802.27	
100	No	96.15%	6,712.27	25.32%
	Yes	95.91%	5,012.39	

Tables 6 and 7 show the benefit of integrating the trust notion into a Minmin-based RMS. Table 6 was run for the inconsistent LoLo heterogeneity with 5 machines. In Table 6, the completion time was reduced by almost 24%. Table 7 was run for the consistent LoLo heterogeneity with 5 machines. In Table 7, the completion time was reduced by almost 26%.

Tables 8 and 9 show the benefit of integrating the trust notion into a Sufferage-based RMS. Table 8 was run for the inconsistent LoLo heterogeneity with 5 machines. In Table 8, the completion time was reduced by almost 40%.

Table 8. Comparison of average completion time for inconsistent LoLo heterogeneity using the Sufferage heuristic.

# of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement
50	No	92.59%	5,257.31	39.66%
	Yes	93.96%	3,172.09	
100	No	96.60%	9,609.78	38.40%
	Yes	97.08%	5,919.49	

Table 9. Comparison of average completion time for consistent LoLo heterogeneity using the Sufferage heuristic.

# of tasks	Using trust	Machine utilization	Ave. completion time (sec)	Improvement
50	No	94.14%	4,473.05	32.67%
	Yes	95.32%	3,011.81	
100	No	97.11%	8,356.33	33.19%
	Yes	97.33%	5,582.56	

Table 9 was run for the consistent LoLo heterogeneity with 5 machines. In Table 9, the completion time was reduced by almost 33%.

In summary, the simulation results indicate that incorporating trust into resource management heuristics can improve the overall quality of the schedules obtained by the resource allocation process.

6. Related Work

To the best of our knowledge, no existing literature directly addresses the issues of trust aware resource management. In this section,we examine several papers that examine issues that are peripherally related.

In [6], a security architecture for a Grid system is designed and implemented in the context of the Globus system. In [6] the security policy focuses on authentication and a framework to implement this policy have been proposed.

A design and implementation of a secure service discovery service (SDS) is presented in [2]. SDS can be used by service providers as well as clients. Service providers use SDS to advertise their services that are available or already running while clients use SDS to discover these services.

A model for supporting trust based on experience and reputation is proposed in [1]. This trust-based model allows entities to decide which other entities are trustworthy and also allows entities to tune their understanding of another entity's recommendations.

A survey of trust in Internet applications is presented in [8] and as part of this work a policy specification lan-

guage called Ponder [3] was developed. Ponder can be used to define authorization and security management policies. Ponder is being extended Ponder to allow for more abstract and potentially complex trust relationships between entities across organizational domains.

7. Conclusions

Resource management is the central part Grid computing system. In a large-scale wide-area system such the Grid, security is a prime concern. One approach is to be conservative and implement techniques such as sandboxing, encryption, and other access control mechanisms on all elements of the Grid. However, the overhead caused by such a design may negate the advantages of Grid computing. This study examines the integration of the notion of "trust" into resource management such that the allocation process is aware of the security implications. In this paper we presented three scheduling heuristics that incorporated the trust notion while scheduling the resource requests. The performance evaluation involved two phases: (a) determining the overhead in securing common operations and (b) performing simulations to evaluate the benefit of incorporating trust in the scheduling heuristics.

The experiments performed to evaluate the overhead of securing remote computation indicate that the overhead is significant and techniques for minimizing such overhead by eliminating redundant application of secure operations can greatly enhance the overall performance.

The simulations performed to evaluate the effectiveness of the modifications indicate that the performance can be improved by about 40%. Several further issues remain to be addressed before the trust notion can be included in practical RMSs. Some of these include techniques for managing and evolving trust in a large-scale distributed system, and mechanisms for determining trust values from ongoing transactions.

Acknowledgement

A preliminary version of this paper appeared in the *First IEEE International Workshop on Security and Grid Computing*.

References

[1] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," *Hawaii Int'l Conference on System Sciences*, Jan. 2000.

[2] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," *5th Annual Int'l Conference on Mobile Computing and Networks (Mobicom '99)*, 1999.

[3] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder policy specification language," *Workshop on Policies for Distributed Systems and Networks*, 2001.

[4] U. Erlingsson and F. B. Schneider, "SASI enforcement of security policies: A retrospective," *New Security Paradigms Workshop*, 1999.

[5] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations," *Int'l Journal on Supercomputer Applications*, 2001.

[6] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational Grids," *ACM Conference on Computers and Security*, 1998, pp. 83–91.

[7] I. Foster, A. Roy, and V. Sander, "A quality of service architecture that combines resource reservation and application adaptation," *8th Int'l Workshop on Quality of Service (IWQoS '00)*, June 2000.

[8] T. Grandison and M. Sloman, "A survey of trust in Internet applications," *IEEE Communications Surveys & Tutorials*, Vol. 3, No. 4, 2000.

[9] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of Grid resource management systems," *Software Practice and Experiance*, Vol. 32, No. 2, Feb 2002, pp. 135–164.

[10] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107–131.

[11] M. Maheswaran, "Quality of service driven resource management algorithms for network computing," *1999 Int'l Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA '99)*, June 1999, pp. 1090–1096.

[12] B. Misztal, "Trust in modern societies," *Polity Press, Cambridge MA*, Polity Press, Cambridge MA, 1996.

[13] C. Small and M. Seltzer, "MiSFIT: A tool for constructing safe extensible C++ systems," *IEEE-Concurrency*, Vol. 6, No. 3, 1998, pp. 33–41.