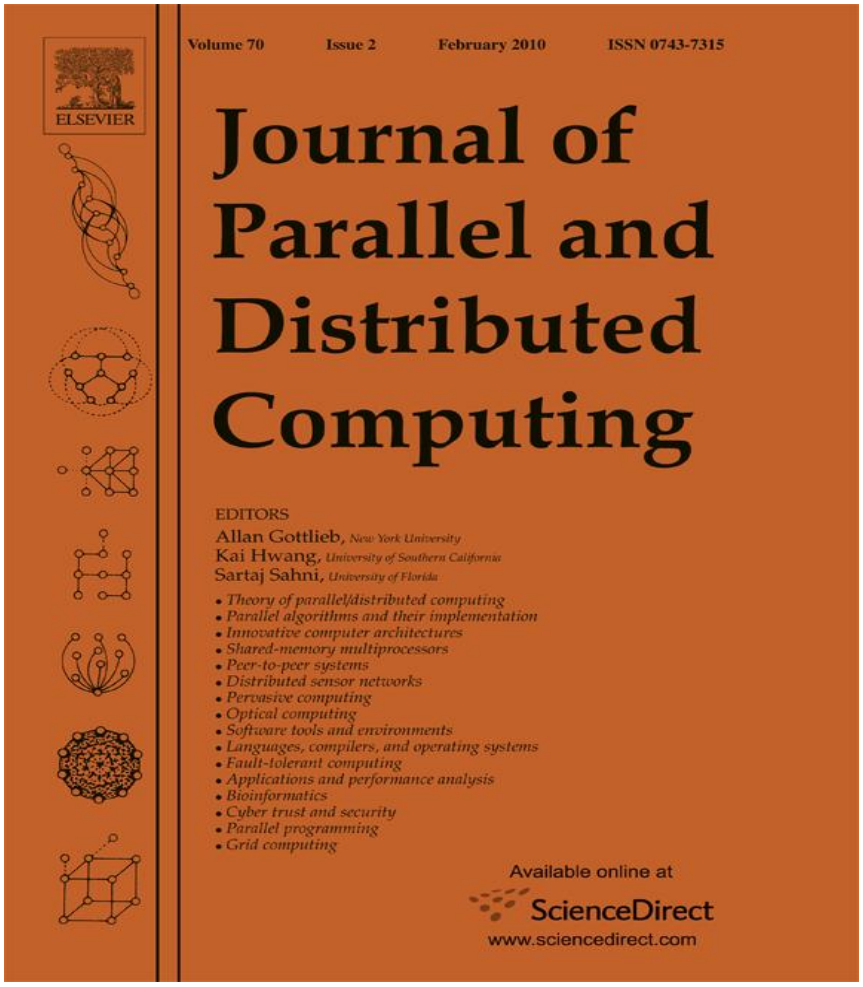


J. Parallel Distrib. Comput. 70 (2010) 111–125

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A reputation-driven scheduler for autonomic and sustainable resource sharing in Grid computing

Zhengqiang Liang [a], Weisong Shi [b,*]

[a] Mobile and Internet Systems Lab., Department of Computer Science, Wayne State University, 407 State Hall, 5143 Cass Ave., Detroit, MI 48202, United States
[b] Mobile and Internet Systems Lab., Department of Computer Science, Wayne State University, 420 State Hall, 5143 Cass Ave., Detroit, MI 48202, United States

ARTICLE INFO

Article history:
Received 23 July 2008
Received in revised form
7 December 2008
Accepted 1 May 2009
Available online 14 May 2009

Keywords:
Resource scheduling
Grid computing
Reputation
Scheduler
Automatic
Trust model
Economic model

ABSTRACT

The obstacle for the Grid to be prevalent is the difficulty in using, configuring and maintaining it, which needs excessive IT knowledge, workload, and human intervention. At the same time, inter-operation amongst Grids is on track. To be the core of Grid systems, the resource management must be autonomic and inter-operational to be sustainable for future Grid computing. For this purpose, we introduce **HOURS**, a reputation-driven economic framework for Grid resource management. **HOURS** is designed to tackle the difficulty of automatic rescheduling, self-protection, incentives, heterogeneous resource sharing, reservation, and SLA in Grid computing. In this paper, we focus on designing a reputation-based resource scheduler, and use emulation to test its performance with real job traces and node failure traces. To describe the **HOURS** framework completely, a preliminary multiple-currency-based economic model is also introduced in this paper, with which future extension and improvement can be easily integrated into the framework. The results demonstrate that our scheduler can reduce the job failure rate significantly, and the average number of job resubmissions, which is the most important metric in this paper that affects the system performance and resource utilization from the perspective of users, can be reduced from 3.82 to 0.70 compared to simple sequence resource selection.

1. Introduction

Grid computing has been aggressively expanded with the increasing demand for high performance computing, with the steady price decrease of the hardware, and the growth of related software support (Globus [21], condor [18,34,55]). After a decade of development, many Grid systems are in use, such as the TeraGrid [54] and EGEE [17]. Although Grid systems are blooming, some Grid users still feel that worldwide Grids are too complex for conventional management approaches to be effective, and too difficult to use and maintain. Using and managing the Grid systems require excessive IT knowledge, workload, and human intervention. The solution is to introduce autonomy into all stages of Grid computing. At the same time, the scale of application is increasing, so that one single Grid cannot handle one large application. Although many Grid platforms are available, most of them are independent without any collaboration, which is detrimental to the utilization of the resource residing in those Grids. With the increase of resource requirement for applications running in a Grid, we are envisioning that, in the foreseeable future, a Grid will step back and act as a PC, and the era of Grid interconnection is coming. So inter-operation is the trend of Grid computing. The emergence of the TeraGrid and OSG [39] has validated that more Grid vendors have launched their efforts to provide Grid inter-operability to allow more Grid resources to be federated and collaborative, to deal with larger jobs and improve the efficiency of resource usage. So for Grid systems to succeed (i.e., to become easy to be used and managed, and to collaborate to finish larger applications), autonomy and inter-operation are necessary in the design of future Grid systems. An autonomic system, at a minimum, needs to be self-configuring, self-healing, self-optimizing, and self-protecting [40]. For Grid inter-operation, the resource heterogeneousness, security, and incentive issues need to be addressed. Resource management, as the core of the Grid systems, is the most important part in the design of a sustainable Grid system. If the supporting resource management scheme cannot keep pace with the Grid's evolvement, a bottleneck of Grid computing will be easily formed. We argue that the one-fits-all and sustainable solution is autonomic and inter-operational Grid resource management.

We propose **HOURS**, a reputation-driven economic framework whose long-term goal is to introduce autonomic resource management and inter-operation in Grid systems. There are two layered components inside **HOURS**—an adaptive personalized trust model (*a***PET**) at the bottom to provide the reputation by quantifying the

* Corresponding address: Wayne State University, 431 State Hall, 5143 Cass Ave, 48202 Detroit, MI, United States.
E-mail addresses: sean@wayne.edu (Z. Liang), weisong@wayne.edu (W. Shi).

0743-7315/$ – see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2009.05.001

quality of sites, and a multi-currency-based economic model on the top to abstract the resource and resource exchange. The trust model coupled with the flexible currency model results in a novel approach for resource management for the next generation of Grid computing. We are not seeking a dramatically subverted way to change the present status of Grid systems, but to deploy our approach incrementally in current Grid systems. **HOURS** aims to be a five-year project, whose objectives are seven-fold:

1. ***Automatic rescheduling***. Automatic rescheduling is very important to improve the productivity for large and long-time jobs. The running of the emulation in this paper is one of the best examples. The emulation is executed under a Linux environment where Cron (a system management tool) is running in each of 11 selected machines. The jobs running on these 11 machines are tightly coupled. When the emulation finishes 50%, after 12 h of running, one machine reboots automatically because of Cron's regular maintenance scheme. This leads to the failure of the entire emulation and wastes the previous 12 h of work. The same happens again and again until we figure out Cron's existence and disable its reboot option. For applications similar to our emulation, it is extremely beneficial if auto-rescheduling for a failed part of jobs can be rescheduled and re-run automatically and transparently in other machines, which can improve the productivity significantly.
2. ***Security and robustness***. The current Grid environment mainly relies on firewalls, complicated configurations, and administrator's interventions for system security and management. This hinders the growing future of Grid platforms, where smaller Grid systems from different Virtual Organizations (*VO*) join together to share the resources. **HOURS** introduces security and robustness for both the user and administrator through its embedded monitor function provided by the reputation and currency information. Malicious/misbehaving sites/nodes (with low reputation) can be automatically excluded from the system; the system can transparently resubmit or migrate the job for users by exchanging and redeeming for the same currency as the previous submission with other nodes.
3. ***Optimal/suboptimal resource scheduling***. Optimal/suboptimal resource scheduling is a traditional research topic. It is still challenging and important in future Grid platforms, and directly related to the fine grain performance of the Grid systems.
4. ***Incentives introduction***. The future Grid system will be more open and Peer-to-Peer (P2P)-like. A good incentive mechanism is needed to enforce nodes to cooperatively contribute resources and use resources.
5. ***Heterogeneous resource sharing***. In **HOURS**, each resource is represented by one currency. The goal of heterogeneous resource sharing can be easily achieved by introducing an exchange rate among different currencies. Different Grid platforms can unite easily through the inter-Grid currency exchange.
6. ***Resource reservation***. In **HOURS**, the concept of resource reservation is equal to the currency holding. A site can use resources from other sites at any time by redeeming the other sites' currency owned by the requesting site.
7. ***Service level agreements (SLA)***. The pricing mechanism in the currency model can support SLA by specifying the price for services with different qualities.

As the first step of the **HOURS** project, we intend to design and implement a reputation-based resource scheduler together with a simple currency model for resource management which provides **incentive-compatibility and self-protection**, and improves the productivity of current Grid systems by **reducing the number of task/job**[1] **resubmissions**. As we may see in the remainder of the paper, a well designed resource scheduler has deserved a paper to describe and evaluate, so the resource scheduler is the main focus in this paper. But for the completeness of the **HOURS** framework, a simple currency model will also be involved. There are a lot of factors in the resource scheduling influencing the Grid's productivity, for example, the quantity of resources and the efficiency of the scheduling software. Grid owners can enhance the Grid hardware processing capacity by investing more in hardware purchases. However, the most preferable and economic option is to design an efficient resource scheduler to explore the Grid's maximum processing potentials. A deficient resource scheduling software design can lead to long waiting time, huge slowdown, and low throughput. But currently the major concern for the users, most of who are not from computer science, is their frequent intervention for program running, e.g., manual resubmission after the job failure; for the system administrators, their concern is how to reduce the frequent interventions for maintenance. These have replaced the traditional views (like slowdown) to be the biggest obstacle for the productivity of Grid computing. In this paper we design and evaluate a reasonable and extensible resource scheduler in **HOURS** to reduce the interventions from the Grid users and administrators.

The rest of the paper is organized as follow. We first categorize the applications running on the Grid from two orthogonal perspectives. We then describe the **HOURS** framework in Section 2. In Section 3 we describe the experiment methodology and the metrics. The experiment results are given in Section 4, followed by the description of the related work in Section 5. Finally, we conclude the paper and then describe the future work in Section 6.

1.1. Application categories

Grid computing has been applied in many fields, and the applications running in the Grid have more diversity than before. In this section, we try to divide these applications into different categories in a 2-D space based on the grain of scheduling and the strictness of completeness. The clarification of applications is helpful for us to understand the behavior of applications and its corresponding resource scheduling approach. First there are two kinds of resource scheduling grains.

1. **Micro-scheduling**: Some applications can be spilt into multiple pieces of independent jobs to run in the Grid without any precedence and dependency. For these applications, resource scheduling has a lot of flexibility by assigning jobs to any site at any time. A typical example is SETI@HOME [3].
2. **Macro-scheduling**: In this category, applications have to be restarted from the beginning if one or more jobs fail, because the jobs of an application are tightly coupled. For example, most applications written in the shared-memory programming model belong to this category.

On the other hand, considering the cost and time there are some applications which do not require completing the running after getting acceptable results. So we have two types of Grid applications.

1. **Application with strict completeness**: This type of application must be completely finished to get the final result. Even if a job has an error, the result may be completely wrong.
2. **Application with loose completeness**: The result precision of the this type of application is incrementally improved as the progress of running takes place. For some applications, like Web

[1] In this paper, a task is defined as the whole application, and a job is a logic running unit of the task after the task is split.

Complete Requirements	Micro-Scheduling	Macro-Scheduling
Strict Completeness	Work Flow Applications	Single Thread/Process Applications
Loose Completeness	SETI@Home [3], Web Search, Graph Render [5]	Emulation in this paper

Fig. 1. Classifications of Grid applications [3,5].

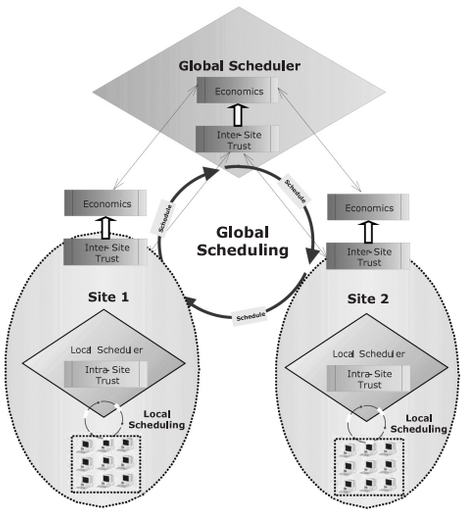


Fig. 2. System architecture.

search and graph rendering, they will have a long cost and time as the application running approaches being finished. To save money and time, this application can stop in the middle of execution after getting acceptable results.

Fig. 1 summarizes the application categories. In the Section 3, we will evaluate the performance of our scheduler with macro- and micro-scheduling, under the background of strict completeness. In the future, we will make **HOURS** compatible with loose completeness applications by considering their special characteristics. It is worth noting that the emulation in this paper is a loose-completeness application since we can see the rough result by finishing say 90% of the trace.

2. HOURS design

In **HOURS**, the reputation is supported and quantified by the trust model. In the rest of the paper, we will use the notion of reputation and trustworthiness interchangeably. The major objective of **HOURS** is to introduce both autonomy and interoperation into the resource management by building a dependable trust-based trading model as a substrate infrastructure for resource management in the Grid. We envision that combining the trust model and economic model will provide a solid foundation for efficient resource scheduling and system management. Trust information will be used to differentiate the sites/nodes with different qualities. The economic model built on top of the trust model provides an effective mechanism for higher-level resource sharing amongst sites. In this paper, we mainly focus on the design of a reputation-based scheduler. Although in this paper the reputation-based scheduler is introduced within the framework of **HOURS**, it can be an independent Grid co-scheduler and also can be applied in other frameworks where a Grid scheduler is needed. The introduction of a simple currency model in this paper is for completeness of the description of the **HOURS** framework and to lay a solid foundation for future improvement and development.

The overall system architecture is depicted in Fig. 2, organized as a hierarchical structure with the following concepts:

- **Scheduler**: There is a *global scheduler* (G-Scheduler) taking care of the resource scheduling amongst sites. When a job dispatched from the G-Scheduler reaches the site level, a *local scheduler* (L-Scheduler) assigns the job to run in a set of nodes based on its local scheduling decision. In our design, we limit the workload of the G-Scheduler to the minimum amount to make it scalable. The queues holding waiting jobs of the G-Scheduler and L-Scheduler are called the G-Queue and L-Queue respectively, both of which are First In First Out (FIFO) queues. When rescheduling is considered, a job which has failed due to the shortage of resources will be removed from the beginning and put to the end of FIFO queue again.
- **Resource Scheduling**: Corresponding to two hierarchical schedulers, there are two kinds of resource scheduling: *global (site) scheduling*, which is executed by the G-Scheduler to select running sites, and *local (node) scheduling*, which is executed by the L-Scheduler to select running nodes. In our approach, reputation-based resource scheduling, the resources (sites from the angle of the G-Scheduler, or nodes from the angle of the L-Scheduler) are scheduling based on the trust value. For comparison, a sequential resource scheduling is introduced, where resources (sites or nodes) are ordered and selected sequentially based on their ID. The reason for choosing sequential resource scheduling as the comparison baseline is that it is used generally in existing practical systems.
- **Trust (Reputation)**: Trust (Reputation) information is used for both global and local scheduling to improve the hit of job dispatching to good-quality sites/nodes. *Inter-site trust* is used in global scheduling to select sites, and *intra-site trust* is used by the L-Scheduler to select nodes. In the following, trust and reputation will be used interchangeably.

Sites and nodes are two system entities in the system hierarchy. Each site represents a set of nodes and a node is a logical unit within a site. This is a logical site–node concept. In real systems, a site may have several clusters or sites. In this case, the site–node concept can be recursively extended so that a node stands for a cluster/site with a local scheduler. The hierarchical structure then has the corresponding extension. For simplicity, in this paper, we limit the discussion to a two-level system, i.e., the node stands for a physical CPU/machine. The basic system running procedure is then described as follows:

1. ***Task submission***: A task is submitted from the submission site to the G-Scheduler; all tasks will stay in the G-Queue.
2. ***Global scheduling***: The G-Scheduler schedules the task with First Come, First Serve (FCFS) policy from the G-Queue, and then selects the running sites based on the inter-site trust information (or sequential selection if without a reputation mechanism), and dispatches the job to the selected sites meeting the scheduling requirements. At the same time, the G-Scheduler changes the stock of currency between the submission site and selected running sites based on the reputation information and resource request. The selected running sites will put the job assignments from the G-Scheduler into the L-Queue.
3. ***Local scheduling***: The L-Scheduler in the running site also schedules jobs from the L-Queue with FCFS, makes node selection based on the intra-site trust information (or sequential selection if without a reputation mechanism), and dispatches the job to the selected running nodes.

4. ***Job running***: Jobs will be running in the selected set of nodes with local scheduling. A node can either finish the job successfully or fail the job running because of node unavailability. If the system can support auto-rescheduling of failed jobs, the failed jobs can be rescheduled to run in other nodes in the same site for a certain number of times. The job is considered as failed only after it still cannot be finished after the allowed maximum number of reschedulings.
5. ***Running result report***: The running results with the number of successful and failed jobs will be reported to the G-Scheduler after the running site stops the job running (either finished or failed). At the same time the site updates the intra-site trust based on the running result.
6. ***Inter-site trust update***: The G-Scheduler updates the inter-site trust based on the reported running results (or perceived running results if the G-Scheduler owns the job running monitor function). If rescheduling is allowed, the failed jobs will enter the G-Queue again, and wait for the next global scheduling. This procedure continues until all jobs in one task have been successfully completed.

As the main component of **HOURS**, the reputation-based scheduler is general enough in the Grid scheduling and can be operated independently or easily integrated in other Grid middleware.

2.1. Trust hierarchy

In **HOURS**, each site issues corresponding amounts of currency according to their contributed resources (i.e., a site can control its local autonomy by issuing its own currency.) The resource sharing is implemented by currency exchange and currency redemption. We relate the currency exchange ratio to the site's trustworthiness T, so that the site's resource quality can be differentiated. The trust model to calculate the site trustworthiness is an adaptive personalized trust model (*a***PET**).

The trust model has been widely applied in the dependable distributed computing environment. The trust hierarchy, inter-site trust and intra-site trust are the core of the whole **HOURS** framework. The purpose of introducing reputation-based scheduling in this paper is three-fold:

- *Increasing the hit of a good-quality site/node for resource scheduling.*
- *Providing self-protection and self-organization for both the node and site level.*
- *Supporting the upper economic model.*

The inter-site trust and intra-site trust are calculated with two trust models with different complexity and implementation cost. The reason we introduce different trust models in the trust hierarchy is to find the best trade-off between implementation cost and preciseness. The scheduling and management for sites is more complicated than the inside of the sites. So the inter-site trust is derived from a novel and comprehensive trust model *a***PET**, while the intra-site trust is calculated by a simpler history-based approach.

Inter-site trust calculation: *a*PET: *a*daptive PErsonalized Trust model. In *a***PET**, the trust is defined as *the subjective probability by which a site, A, expects that another site, B, performs a given action as well as expected in a certain time*. In *a***PET**, the trustworthiness, T, is derived from two parts: self-experience information, E, achieved through direct interactions with other sites, and rating from others R. T is calculated as $T = \alpha * R + (1 - \alpha) * E$. Self-experience is regarded as a kind of reliable source in the derivation of the trustworthiness value. However, this kind of information cannot bring efficiency to the trust model. The rating from other sites is introduced for efficiency purposes, which can help to discover

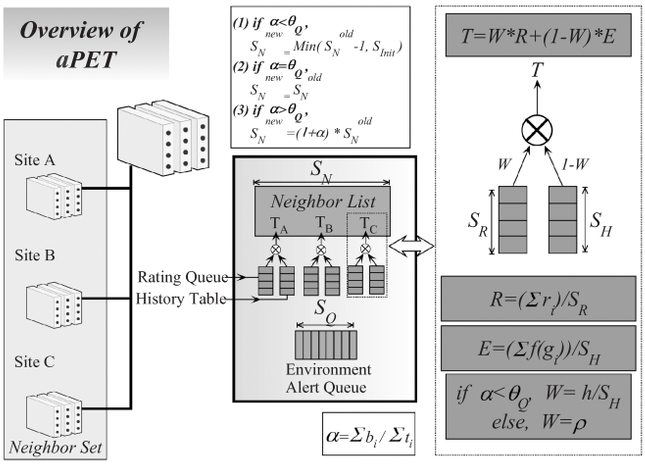


Fig. 3. An overview of the *a***PET** model.

the quality of other sites even without direct interactions. But rating is not reliable due to the dynamic changing environment and different experience of sites. *a***PET** integrates these two kinds of information with dynamic weight to show advantages while inhibiting disadvantages. Actually, *a***PET** is built on our previous *PET* model [32] and our thorough analysis to the main rating models in current researches [33]. *a***PET** is distinguished in its ability to self-adaptively change the weight for trustworthiness derivation according to the change of the environment and the richness of self-experience of sites.

Fig. 3 shows an overview of *a***PET**. Every site has its own neighbor set. The neighbor set is stored in the *neighbor list*, which is a global data structure in *a***PET**. In Fig. 3, a site has three neighbors, sites A, B, and C, as shown in the left of Fig. 3. Correspondingly there are three elements in its neighbor list, each of which includes the fields of site ID and their trustworthiness values. For every neighbor, two local data structures, the *rating queue* and the *history table*, are used to store the rating and interact-derived information respectively. The global data structure, the *environment alert queue*, is employed to sense the severity of the environment. The neighbor list, rating queue, history table, and environment alert queue are all FIFO queues. Their sizes are denoted as S_N, S_R, S_H, and S_Q, respectively.

Our previous research shows that it is not worth paying too much attention to the rating aggregation algorithm considering the cost and the payback [33]. In *a***PET**, the simple average scheme is used to aggregate the ratings. The rating, r_i, is the ith element in the rating queue, whose value is from 0 to 1. The calculation of self-experience E is based on the site average **job success rate** $f(g_i)$ ($f(g_i)$ is the job success rate in one result report ***i*** from the site) within a time window (S_H is the window size). Suppose site A's trustworthiness value in site B is $T_{A\_in\_B}$, then site B's rating r regarding A, $R_{A\_in\_B}$, is equal to $T_{A\_in\_B}$. Since E stands for the reliable information, it deserves more considerations. The adaptiveness of *a***PET** is mainly embodied in its capability to self-adjust the weight, W, and the size of the neighbor list, S_N, according to the severity of the environment. There is one important metric, the environment-aware factor, α, to guide the adaptiveness. The environment alert queue is used to sense any change in the surrounding environment, which records the recent running results from all neighbor sites. Each element of this queue is a vector (b_i, t_i), where b_i is the number of failed jobs among the total t_i jobs in one result report ***i***. α is defined as the proportion of failed jobs in the most recent interval. A large α indicates that the environment is bad (the current neighbor set is not in a good status to run the jobs). In this case, it is better to run the jobs in a new set of neighbors. The way is to increase S_N, because the larger the neighbor list is,

the larger the probability that a good-quality site can be included in the neighbor set. However, increasing the size of the neighbor list can incur significant storage cost (each additional element in the neighbor list will lead to the installment of one rating queue and one history table). Moreover, it can bring more network traffic when the number of objects to be rated increases. We define a severe threshold, θ_Q. When $\alpha > \theta_Q$, the environment is thought to be severe so that the new size of the neighbor set S_N^{new} will be enlarged to $(1+\alpha) * S_N^{old}$. When $\alpha = \theta_Q$, which means the healthiness of the environment is moderate, the neighbor list keeps the same size as before. When the environment turns good, as implied by $\alpha < \theta_Q$, the neighbor list will be shrunk to $\text{Min}(S_N^{old} - 1, S_{Init})$ to reduce the cost of storage and traffic, where S_{Init} is the initial size of the neighbor list. The research in [32] shows that decreasing the weight of a rating is useful to inhibit the negative effect of bad ratings. So in *a***PET** when $\alpha \geq \theta_Q$, the weight of rating W is set to a fixed low value ρ. Simulation results in [32] suggest that if ρ is set to a value between 0.2 and 0.3, the negative effect of the rating can be greatly inhibited with an acceptable efficiency sacrifice. If $\alpha < \theta_Q$, the environment is healthy, and then W is adjusted according to the quantity of the self-experience information. In this case, W is defined as the injection degree on the self-experience, i.e., the ratio of the number of interactions with the neighbor h to the size of the history table S_H in a specific time. It is worth noting that, for every small time interval, the oldest element in the history table will be deleted to refresh the self-experience information. θ_Q is an important parameter for the adaptiveness. Finding an efficient way to determine its value is worthy of deeper study.

Intra-site trust calculation: History-based Trust Model Generally speaking, the node management inside a site is rather simpler than amongst sites, because all nodes are under the same administrative domain with a similar usage and management policy. So for the intra-site trust calculation, a much simpler trust model, i.e., *a history-based trust model*, is used, where the node trustworthiness is defined as the average **job success rate** $AVG(f(g))$ in the history. To make the model sensitive and fast to catch the node failure, in some cases (macro-scheduling or without rescheduling, explained in Section 3.1) the wrecker nodes (nodes fail to finish the jobs so that even other nodes in the same site can finish their parts, the site is deemed to run the task or part of the task unsuccessfully from the side of global scheduler) take all responsibility with no trustworthiness degradation for other nodes in the same site.

2.2. Economic model

*a***PET** is designed to allow each site to sense the change of the quality of its neighbors and the environments. This is not adequate, with regard to the introduction of incentives, collaborations, and efficiency for resource sharing. So on top of the trust model, we borrow the idea from real economics to build a multiple-currency-based economic model to complement the weak point of the trust model. Once a site owns currencies from its neighbor, it also has a hard reservation for the resources represented by these currencies. So in our approach, resource sharing is presented as a form of currency exchange. The idea of hard reservation is different from the soft reservation in SHARP [19]. The advantage of soft reservation lies in the efficiency improvement for the resource usage. However it has numerous disadvantages.

- It cannot precisely describe the snapshot of the resource requests, so it is difficult to achieve the goal of optimal resource allocation, which needs the precise information about the resource availability.
- It is hard to predict the resource usage in the next time slot.
- The failure in the resource competition can lead to unbearable delay for some time sensitive applications.

The fundamental underlying concept in our approach is the precise representation of various resources in terms of currency. Currency abstracts the grain, type, and property of resource sharing to support adaptive and optimal resource scheduling. There are five major fields in the currency representations:

1. *Resource Amount*. The absolute amount of resources which the currency represents.
2. *Resource Type*. The type of resource the currency is corresponding to, such as CPU, storage, bandwidth, etc.
3. *Resource Available Period*. This property specifies the available time (vector) of the resource.
4. *Resource Parameters*. These specify the properties of resources, such as CPU frequency, storage capacity, etc.
5. *Signature*. Signature is used to sign the currency so as to make it verifiable and non-repudiated.

Currency is the contract for the resource reservation and it stands for the agreement of the resource sharing between two collaborative sites. Currency activity only happens in the site level. We regulate the currency exchange ratio with the fluctuation of the trustworthiness value of the sites. A simple currency exchange approach is adopted in this paper: the currency exchange ratio $R_{A\_from\_B} = T_{A\_in\_B}$, where $R_{A\_from\_B}$ is the currency exchange ratio when site A asks for currency exchange from site B, and $T_{A\_in\_B}$ is the trustworthiness value of site A in the eyes of site B. That is, when site A asks for site B's currency, B will return $T_{A\_in\_B}$ units of B's currencies for each unit of A's currency. The basic idea behind this is that the more truthworthy a request site is, the more currencies it can get from the destination site in the exchange procedure. So in this way, each site has incentives to provide good service to maintain a high trustworthiness value. If a site continuously provides poor service (high job failure rate in this paper), it will eventually be evicted from the community because its currency is worthless due to its low trustworthiness. The currency model is not the focus of this paper. However, even using this simple approach, we still can see its potentials from Section 4.7, where a good-quality site can earn more currencies. Based on the concept of currency, we take the commodities market as our design prototype instead of the auction markets. The reason is that an auction market has the problem of high delay and traffic to schedule the resource. Through well articulating, the commodities market can be optimized to be an efficient scheduling approach.

In the computing community, though each site has to collaborate with other sites to build a friendly resource-sharing community, each site also wants to keep the self-management to control the resource sharing, especially when the sites are located in different competitive organizations. Our approach can provide flexible autonomy for each site: a site uses its own policy to decide the amount of resource to be shared by changing the total amount of currency it issues; a site can also decide whether to honor the currency redemption and authorization; a site self-decides the job submission according its own policy. Normally a site rejecting legitimate redemption requests will be punished by having its trustworthiness value lowered. However, in some cases, especially when facing some unexpected malicious attacks, self-management of sites can provide self-protection. So our approach is able to provide the maximum self-management and self-protection even under emergent situation like suffering malicious attack.

A potential problem for multiple currency is how to make each site honestly issue its currency corresponding to its actual amount of resource. A site can arbitrarily issue as many currencies as it can regardless of its actually resource amount. That is why we need the underpinned trust model. A misbehaving site that issues its currency excessively is bound to the result of lowering its service

quality, because its actual service capacity cannot meet the service requests from the sites holding its legitimate currency. In that case, the trustworthiness of the misbehaving site will be lower and it will eventually be evicted from the system if it continues its misbehavior, as explained above. Since the currency model is not the main focus of this paper, we are not going to extend the discussion on how the currency model defends itself from attacks. More details can be found in our paper [31], which shows the advantages and robustness of the currency model when combined with the trust model.

Extend the economic model in heterogeneous environments From the *Resource Type* field in the currency format, we can see that the economic model is to be used in the heterogeneous environment. The crux of heterogeneous resource sharing is how to set up the proper pricing mechanism to make different resources comparable and exchangeable. A naive approach is to relate the resource price to its corresponding hardware price in the real market. However, the frequent update of the hardware price can introduce a considerable management workload. Moreover, associating the price with the market price cannot reflect the real value of resource under the virtual market in the resource-sharing community, which is negative to build a harmony and healthy virtual economic framework. It needs further consideration for the pricing. A possible approach is to adaptively change the resource price according to the usage information provided by the scheduler. In the next step, a more comprehensive economic model, named **M-CUBE**, will be integrated into **HOURS**. The details of **M-CUBE** can be found in our previous paper [31], where we have more details about the format of currency, the advantages of the model, the advanced currency exchange protocol, and its ability to defend itself against all kinds of attacks including the excessive currency issuing problem.

3. Experimental methodology

In this section, we first describe how to set up the emulation, including how to simplify **HOURS** to fit the emulation for the current Grid and how to set the experiment configurations. After that the metrics for the result measurement of the experiment are described.

3.1. Methodology

We develop an emulator by emulating the current TeraGrid with 11 computing sites in a local area network, where there is one node to take care of the role of the G-Scheduler and 11 nodes to mimic the computing sites. Real job traces and node failure traces are used in the emulation. The description of the complete **HOURS** framework in Section 2 is to be applied in the general and future Grid systems; however, in the initial stage of the **HOURS** project, we decided to implement a simplified version which is also used in the emulation. The highlights of the simplification are described below. Since there are only 11 sites in total, the size of the neighbor list is no longer able to dynamically change but is fixed to be with a size of 11. Thus the adaptiveness of *a***PET** in the emulation mainly focuses on the weight adaptiveness. All inter-site trust and currency information is directly maintained in the G-Scheduler, instead of being frequently updated from sites. To better illustrate the results, we use only one kind of resource – CPU – in the emulation. Finally, in the **HOURS** framework multiple-currency is introduced to serve the purposes of autonomous management amongst sites. To simplify the emulation, we choose a single-currency implementation in the emulation. A global trust table, a history table, and a rating table are maintained for each site. The trustworthiness value is updated by using the *a***PET** model. The job requests in the G-Queue are scheduled with FCFS policy. Once the G-Scheduler gets the running results from the local sites, the history table will be updated. The trustworthiness value of each site in the trust table will be updated with a fixed frequency. If the change of a site's trustworthiness value is larger than a threshold, it will be propagated by the G-Scheduler as ratings and the trust tables of other sites will be updated. Similar to the G-Scheduler, each site has a local scheduler, a local trust table, and a history table, but without the rating table. The trustworthiness value of each node within the site will be updated once the L-Scheduler perceives the running result of the node, by calculating the accumulative running result history. In the local scheduling, the L-Scheduler can select nodes either based on the reputation of nodes or sequence selection policy (without trust).

All machines in the emulation are configured with Intel Pentium 3 1.0 GHz processors, 512 MB ram, 16.9 GB hard drive, and the OS is Linux ES V4.0. Fig. 4 illustrates the overview of the emulator. In the emulation, we totally emulate 11 sites, the same number of sites in the current TeraGrid [54] configuration. To make the emulation close to the real system, we use two real traces for the site emulation: the job trace from San Diego supercomputer center (SDSC) DataStar log, SDSC-DS-2004-1.swf [2,9], and the node failure trace from the Los Alamos National Laboratory, LA-UR-05-7318-failure-data-1996-2005.csv [1]. We choose the SDSC-DS trace because SDSC is one site of the TeraGrid. For the node failure simulation, we do not use a synthetic method to generate the trace according to the node status, e.g., node workload. Instead, we use the real node failure trace, which actually specifies the maximum possible failure time of each node that can be detected. Intuitively, the node failure rate might be related to the load of nodes. However, a recent experience study from CMU by Schroeder and Gibson [46] shows that the failures are actually most related to hardware, software, and operations. Their recent work [47] finds that the failure rate of a system grows proportionally to the number of processor chips in the system, and there is little indication that systems and their hardware get more reliable over time as technology changes. Also, we have not seen any publicly available data/results about how load is related to failure traces. Therefore, based on the current knowledge of failures in the computing community, we think that our assumption is reasonable and practical.

We scale the time of both the job trace and the failure trace so that they can fit together in the emulation which lasts for around one day for each emulation run. We call the trace after scaling the normalized trace. The total number of jobs in the normalized job trace is 13,054, and the average running time is 106.63 s. During the normalization, some small job requests with running time less than the threshold will be deleted from the trace. We notice that after the normalization, the job trace may have different cameralistics from the original one; in particular, it cannot reflect the part of the trace with small jobs. However, if we do not filter out the small jobs, it is very difficult to run the emulation because of the huge number of jobs. When we try to run the emulation without any job filtering out, the job queue is unbearably long, and the running time is extremely long. In that case, the metrics queuing time and slowdown are distorted to depart from the real scenario. Although we cannot guarantee that running the normalize trace can reflect the real situation, we believe it is much closer to the case of real Grid running. Moreover, our major concern is to reduce the rescheduling time, which is primarily meaningful for the long-time jobs. Even after the small jobs are filtered out, the long-time jobs are relatively unchanged compared to the original trace. From this angle, the normalized job trace can still be treated as the real trace and be meaningful as the emulation input. Fig. 5 shows the statistics of the traces with CPU number and average CPU failure rate.

We also consider if auto-rescheduling has happened or not in both the global and local scheduling levels. There are

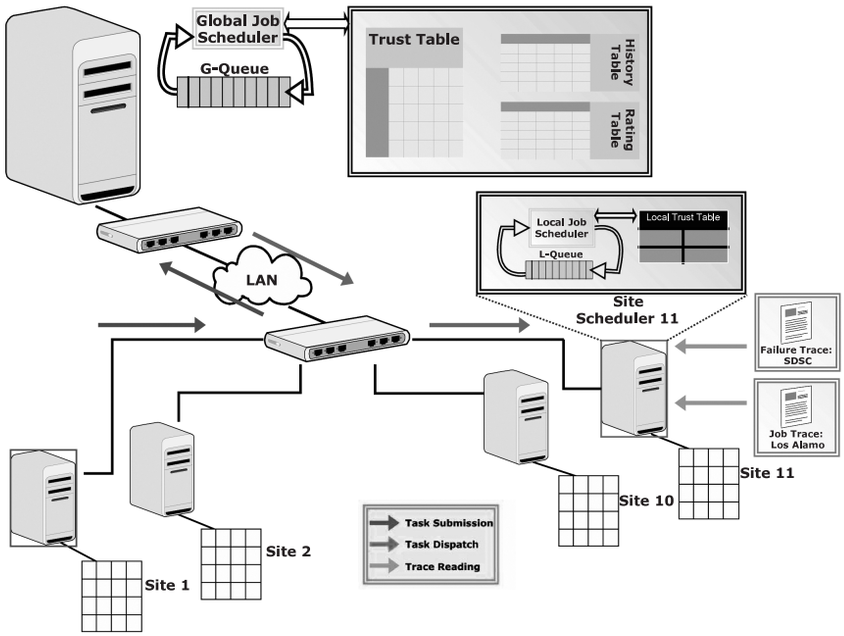
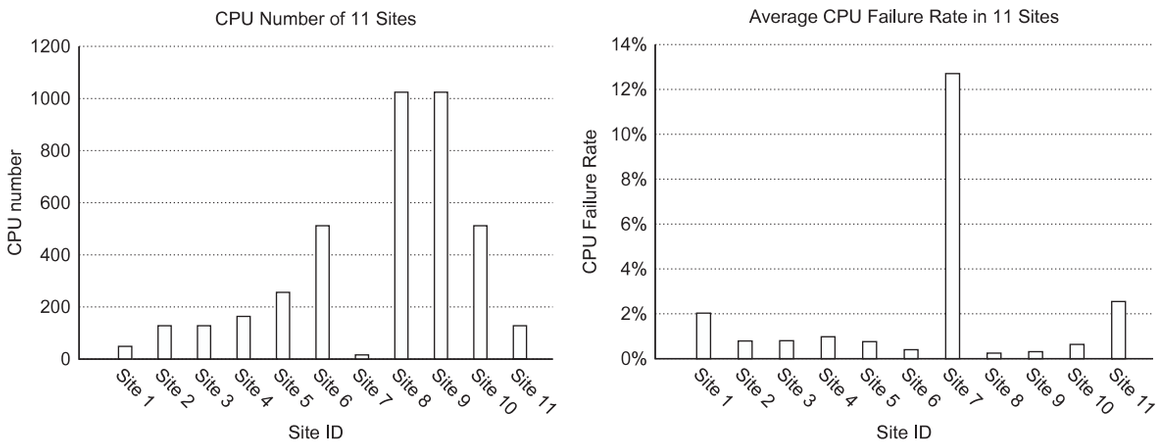


Fig. 4. An overview of the emulator and its deployment.



(a) Number of CPUs in 11 sites.

(b) Site's CPU failure rate.

Fig. 5. Number of CPUs in 11 sites and their average failure rates.

two reasons. (1) With the maturity of the checkpointing and migration techniques [10,11,35,55], together with our currency-based resource representation and reputation-based scheduling, **HOURS** can introduce a highly efficient automated rescheduling, which is promising in the future Grid platform development. It will greatly reduce the intervention of humans for resubmission, so as to improve the productivity of the Grid. (2) We treat the number of reschedulings as the major metric in the performance evaluation. In the emulation, it is challenging, if not impossible, to manually resubmit a job request once it is failed. So automated resubmission is necessary.

Finally, according to different kinds of applications, we also consider two grains of scheduling as explained in Fig. 1:

- **Micro-scheduling**: In the emulation, each application (task) will be split into equivalent pieces of jobs. (In a real application, the job size may be different. In this paper, we will consider only this simple case.) The jobs will be equivalently treated and repeatedly dispatched to multiple sites until all jobs are finished.
- **Macro-scheduling**: To emulate the macro-scheduling, in the emulation all jobs can only be submitted to the sites which have enough resources to run all jobs at one time. For a task if there are no sites which hold enough resources to run all jobs at one time, the task will be moved to the end of G-Queue and wait for subsequent scheduling. If a task is successfully scheduled and runs in one site, one job failing will lead to the failure of the whole task and the task will be resubmitted to the G-Scheduler for rescheduling.

In the emulation, six configurations, as shown in Fig. 6, are compared and evaluated. We can view these configurations from three orthogonal angles, each of which has two cases: micro-scheduling (micro-) or macro-scheduling (macro-), using reputation (-rep) or no reputation (-norep), and rescheduling (-resched) or no rescheduling (-noresched). The six configurations $C0$–$C5$ stand for *micro-norep-noresched*, *micro-rep-noresched*, *micro-norep-resched*, *micro-rep-resched*, *macro-norep-resched*, and *macro-rep-resched*, respectively. In the rest of the paper, abbreviations will be used to refer to a set of configurations. For example, *micro-all-all* stands

Factors \ Configuration #	C0	C1	C2	C3	C4	C5
Rescheduling	No	No	Yes	Yes	Yes	Yes
Reputation Based Scheduling	No	Yes	No	Yes	No	Yes
Micro (MI) or Macro (MA) Scheduling	MI	MI	MI	MI	MA	MA

Fig. 6. The configurations using in the emulation. Each configuration is the combination of three factors shown in the first column of the table.

for $C0$–$C3$, where *all* stands for both cases under the corresponding angle.

3.2. Performance metrics

The major metric to evaluate our scheduler's performance is the number of job reschedulings in the G-Scheduler. We also measure our experiment results with several important traditional metrics. The metrics are defined as follows.

- ***Number of reschedulings***: We define the number of reschedulings of a task as the number of redispatchings from the G-Scheduler.
- ***Failure rate***: For the two scheduling grains, task and job, there are two failure rates: ***task failure rate*** and ***job failure rate***. The task failure rate is defined as the ratio between the number of failed tasks among all tasks submitted from the trace, which is only applied in $C0$ and $C1$ where rescheduling does not apply. For $C2$–$C5$ with a rescheduling mechanism, all tasks will be finished in the end. Similarly, the job failure rate is defined as the ratio between the number of failed jobs among all jobs, which applies in all six configurations.
- ***Queuing time***: This is the total time for the job residing in the G-Queue, which includes the waiting time before getting scheduled and the waiting time for rescheduling.
- ***Site resource utilization***: We define site resource utilization as $\frac{\sum_r t_{busy}}{\sum_r t_{busy}+\sum_r t_{free}}$ to show the resource usage rate, where t_{busy} and t_{free} are the total time in the busy and free stage for each machine in the site, respectively.
- ***Slowdown***: The slowdown (stretch) of a job is the ratio of a job's response time with respect to its runtime on an ideally unloaded system. Since this metric is a compound metric including response time, we are not going to show the result regarding the response time, although we have these data.

4. Experimental results

In this section, we first compare the failure rate, a direct metric to see the performance of scheduler. We then study the amount of job rescheduling, the main metric to show the advantage of our scheduler, under four rescheduling configurations. To understand the effects of reputation in the scheduling, we depict the mapping between the summarized trustworthiness of sites in the G-Scheduler and the site's node unavailability. We also evaluate the performance of the scheduler from the other important metrics including slowdown, job queuing time, and CPU utilization. Finally, to ensure the completeness of our experiments, the economic effects of our preliminary economic model are studied.

4.1. Failure rate

Failure rate is the most direct metric to evaluate the efficiency of the scheduler. In this section, we will take a look at the task failure rate when there is no rescheduling mechanism *micro-all-noresched* ($C0$ and $C1$), and the job failure rate for all six configurations *all-all-all* ($C0$–$C5$).

4.1.1. Task failure rate without rescheduling

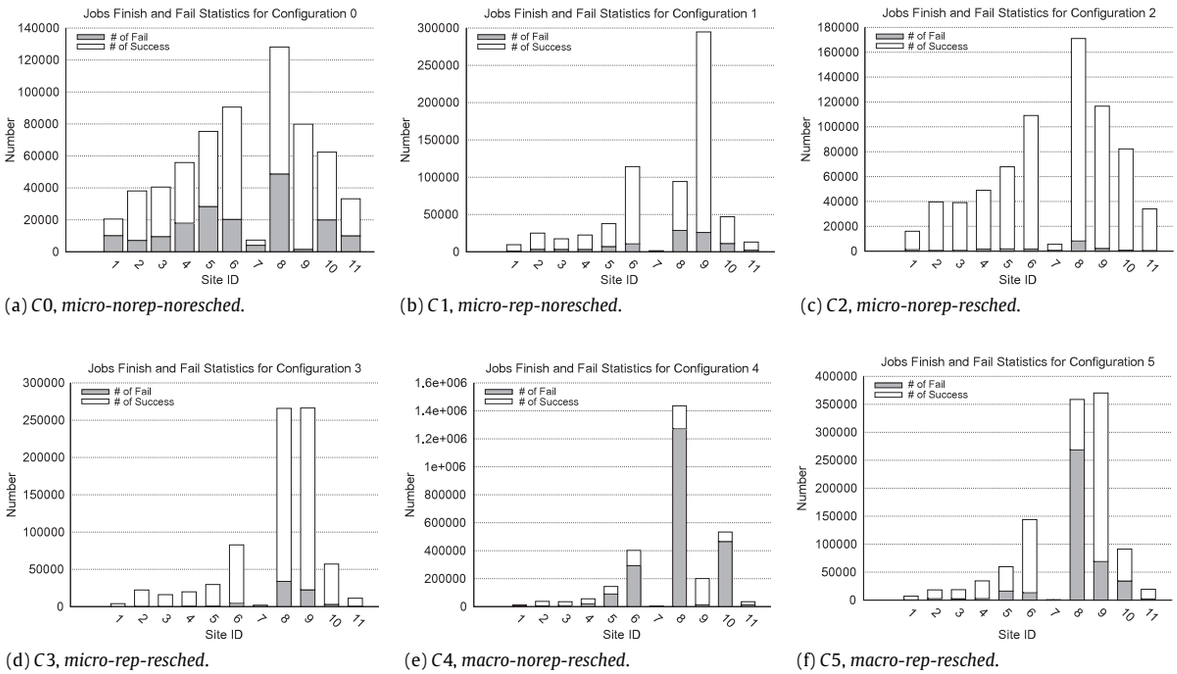
Only for *micro-all-noresched* can we see the task failure rate, since in *all-all-resched* ($C2$–$C5$) all tasks will be finished eventually with the help of rescheduling. With the reputation mechanism in $C1$, among all 13,054 tasks, there are 1536 failed tasks. The task failure rate is 11.77%. Without the reputation mechanism in $C0$, however, the number of failed tasks increases to 3056 (23.41% task failure rate), around two times the task failure rate of $C1$. This shows that the reputation has obviously positive effects in the scheduling to reduce the task failure rate if there is no rescheduling mechanism involved, which is quite common in real deployments.

4.1.2. Job failure rate in 11 sites for all six configurations

Another angle is to use the job failure rate to analyze the efficiency of our approach since jobs have smaller grain than tasks. Fig. 7 shows the number of failed jobs in 11 sites for all six configurations. Under micro-scheduling ($C0$–$C3$), the reputation mechanism makes the sites with high quality attract more workloads. The number of CPUs and the site failure rate are the two major metrics to decide the quality of a site. From Fig. 5, we can see that sites 8 and 9 have the largest number of CPUs, both with 1024 CPUs; sites 6 and 10 have the second largest number of CPUs, both with 512 CPUs. The site failure rates are low ($<1\%$) for all these four sites.

For $C1$, site 9 runs in total around 290,000 jobs, more than twice the job number of the second loaded site 6 (around 120,000 jobs). For $C3$, sites 8 and 9 both run in total around 270,000 jobs, more than triple the job number of the second loaded site 6 (around 70,000). But without the reputation mechanism, i.e., for $C0$ and $C2$, we can see that the difference of workload distribution is not as obvious as for $C1$ and $C3$. For $C0$ and $C2$, the workload distribution is roughly in portion to the CPU number of each site (shown in Fig. 5(a)). This shows that for micro-scheduling, the reputation mechanism is playing an obvious role in directing the workload to the good-quality sites.

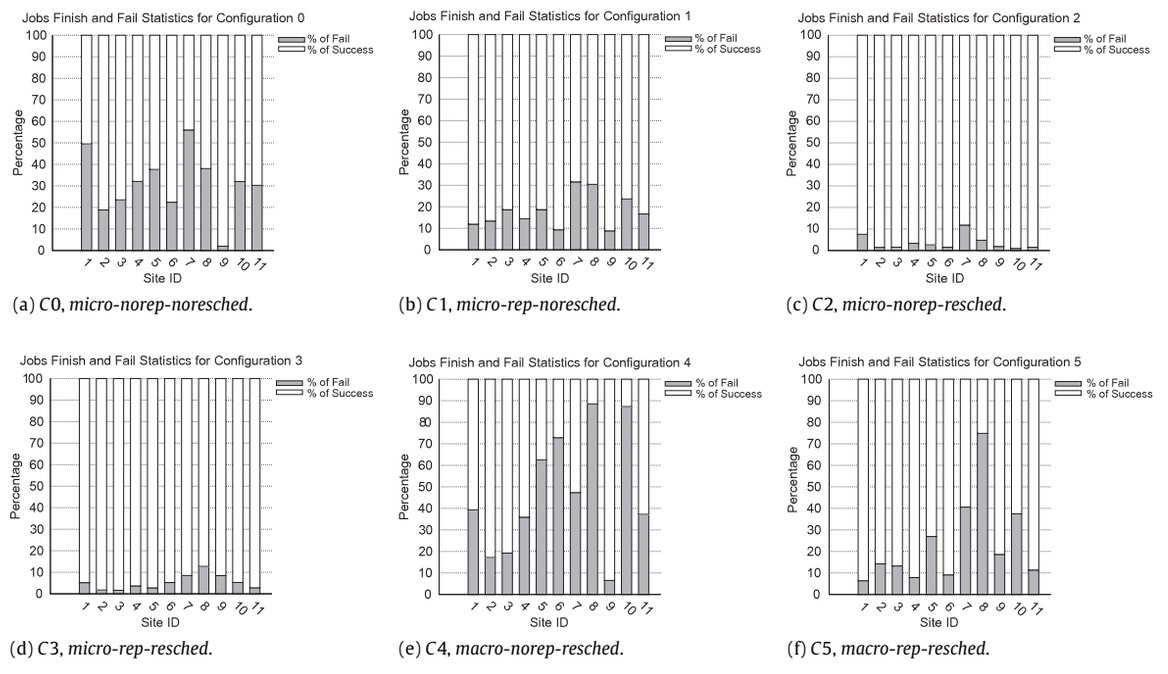
Under macro-scheduling, the workload distribution has a huge diversity amongst sites without the reputation mechanism ($C4$), where we can see that site 8 takes care of the majority of the workload (more than seven times the job number of site 9), and about 90% of its jobs fail. Without reputation mechanism, although site 8 has a large number of failed jobs, it is still chosen more frequently in the sequence site selection since site 8 is in front of site 9. This situation is greatly improved in $C5$, where site 9 takes care of more workload then site 8 because of the direction of the reputation mechanism, and more than 80% jobs are finished successfully. Site 6 also takes care of a considerable amount of workload, and most of the jobs are finished successfully. Fig. 8 shows the job failure rate from the percentage angle, which clearly shows that the overall job failure rate of the system is reduced with the reputation mechanism when we compare $C0$ vs. $C1$, and $C4$ vs. $C5$. $C2$ and $C3$ have a similar job failure rate, and the job failure rate of sites 8 and 9 in $C3$ with the reputation mechanism is even lower than in $C2$ without the reputation mechanism. This shows that when micro-scheduling and rescheduling coexist, the reputation mechanism cannot show too much potential. The reason is that in the local site the rescheduling mechanism exists in $C2$ and $C3$. In the context of rescheduling policy, the job is considered as failed only after it cannot be finished after a certain number of reschedulings. In most situations, jobs can be successfully completed after one local rescheduling even if they fail the first time, regardless of the existence of the reputation mechanism. But currently it is very difficult to implement micro-scheduling together with auto-rescheduling. The performance of $C3$ is still good with the reputation mechanism, although it is not better than $C2$ as expected.



(a) $C0$, *micro-norep-noresched.* (b) $C1$, *micro-rep-noresched.* (c) $C2$, *micro-norep-resched.*

(d) $C3$, *micro-rep-resched.* (e) $C4$, *macro-norep-resched.* (f) $C5$, *macro-rep-resched.*

Fig. 7. Number of failed and successful jobs under six configurations.



(a) $C0$, *micro-norep-noresched.* (b) $C1$, *micro-rep-noresched.* (c) $C2$, *micro-norep-resched.*

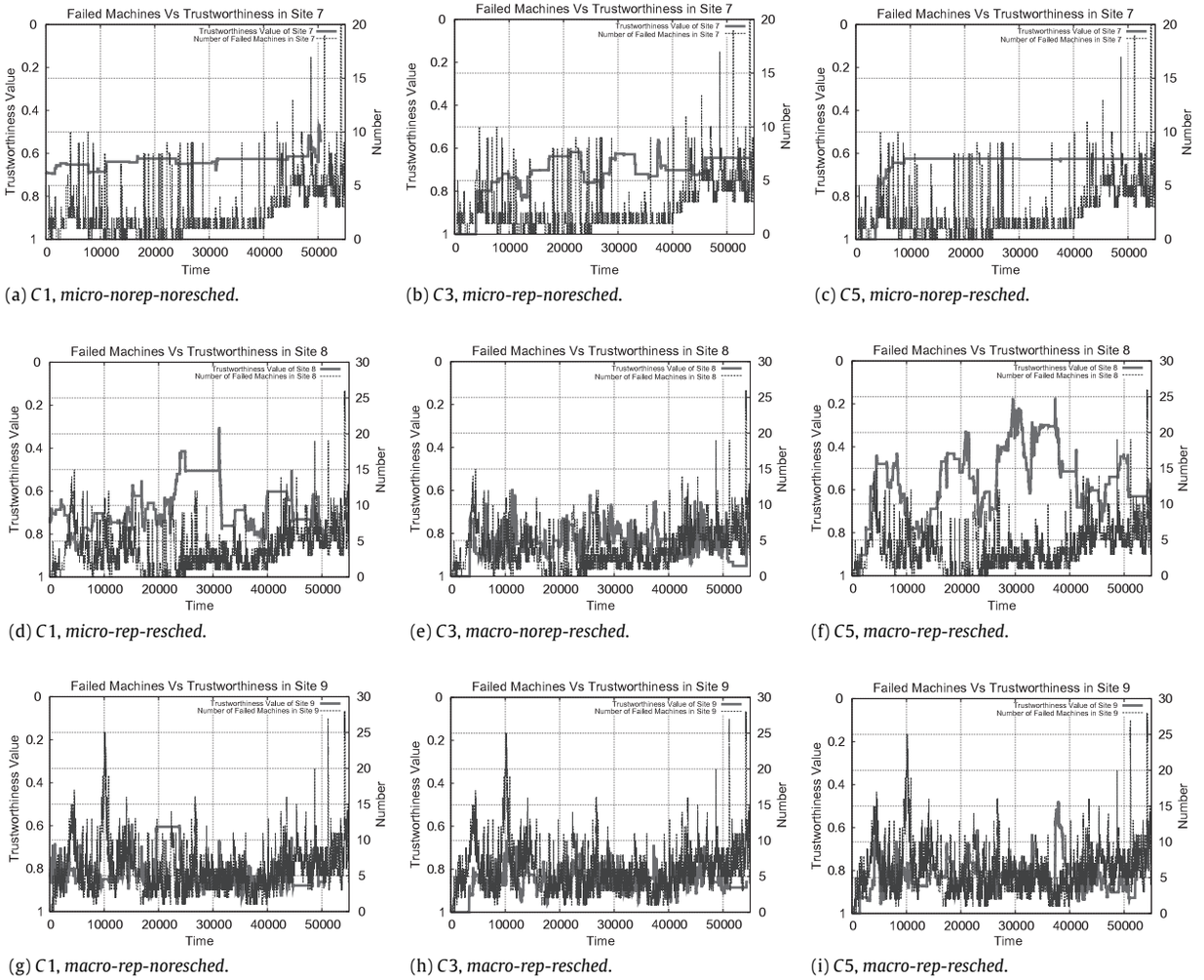
(d) $C3$, *micro-rep-resched.* (e) $C4$, *macro-norep-resched.* (f) $C5$, *macro-rep-resched.*

Fig. 8. Percentage of failed and successful jobs under six configurations.

What we need to discuss further is site 8, with the largest CPU number and low failure rate. It is a good-quality site if only based on the number of resources and CPU failure rate. But its workload is low in $C1$ and the scheduling failure rate is quite high in $C4$ and $C5$. We will investigate the reason from the matching between the site's reputation and its node availability in Section 4.2.

4.2. Reputation vs. site unavailability

Site unavailability is a direct factor that affects the site's reputation, which is defined as the portion of unavailable machines to all machines in the sites. To better understand how the reputation mechanism works, in this section, we will take a look at how the site's reputation catches the site's unavailability. The site's reputation is defined as its average trustworthiness value from the eyes of all other sites in the G-Scheduler. We take three sites as representatives: site 7, which has the lowest CPU number, and sites 8 and 9, which have the largest CPU number, but with different performance. The matching results are illustrated in Fig. 9. From Fig. 9 we can see that site 7's trustworthiness value is not dynamic enough to catch the site unavailability. The reason is that site 7 has the lowest CPU number, so only a very limited number of jobs will be assigned to be run in site 7. So there is not enough information to update its trustworthiness. For sites 8 and 9 with the



(a) $C1$, *micro-norep-noresched.* (b) $C3$, *micro-rep-noresched.* (c) $C5$, *micro-norep-resched.*

(d) $C1$, *micro-rep-resched.* (e) $C3$, *macro-norep-resched.* (f) $C5$, *macro-rep-resched.*

(g) $C1$, *macro-rep-noresched.* (h) $C3$, *macro-rep-resched.* (i) $C5$, *macro-rep-resched.*

Fig. 9. Trustworthiness vs. node failure under configurations $C1$, $C3$ and $C5$. (a)–(c) are the results for site 7, (d)–(e) are the results for site 8, and (g)–(i) are the results for site 9.

largest number of CPUs, there are a large number of jobs running in these two sites. So the trustworthiness values of these two sites changes frequently with rich job running information. For site 8, under $C3$ with rescheduling, we can see the dynamic change of its trustworthiness matches the change of its node unavailability very well, while for $C1$ and $C5$, and especially for $C5$, the match is not precise. But for site 9 with the same CPU number and similar average CPU failure rate, the matchings amongst all three configurations are fitting well. We are not sure what exactly causes this difference. A possible reason is that the adverse combination of job trace and node trace is over the resilience capability of our reputation mechanism in site 8. This is a future research topic. The mismatching for site 8 explains why in Section 4.1 site 8 has a much higher job failure rate than site 9.

4.3. Number of reschedulings

In this section, we are going to see how our approach can reduce the number of reschedulings. Since $C0$ and $C1$ are for non-rescheduling, only $C2$–$C4$ are considered here. Fig. 10 shows the rescheduling statistics for the 1383 tasks with requested CPU number ≥ 128, because the rescheduling mainly happens for large tasks. From Fig. 10, we can see that under micro-scheduling for $C2$ and $C3$, the performance with the reputation mechanism is a little bit better than without the reputation mechanism; the

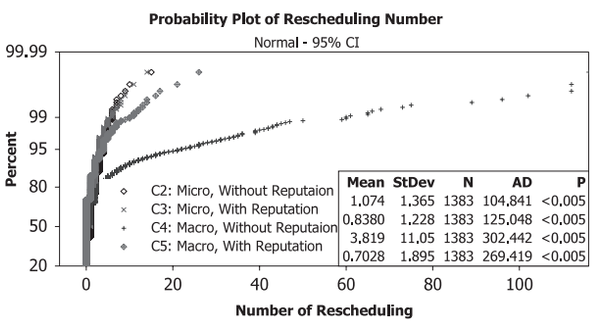


Fig. 10. Number of reschedulings for the job requesting CPU number ≥ 128 under the configurations $C2$, $C3$, $C4$ and $C5$.

average rescheduling number for $C2$ is 1.07, while for $C1$ this number is 0.84. Under macro-scheduling, however, the difference is much larger: 3.82 for $C4$ and only 0.70 for $C5$. With the reputation mechanism, the number of reschedulings under macro-scheduling is less than under micro-scheduling. From Fig. 10, we can also observe that for $C4$ without the reputation mechanism, the rescheduling number can reach around 120, almost 5 times that for $C5$ with the reputation mechanism. This adequately proves the positive effects of the reputation mechanism, because under macro-scheduling, the reputation mechanism is more sensitive, as analyzed in Section 4.1.2.

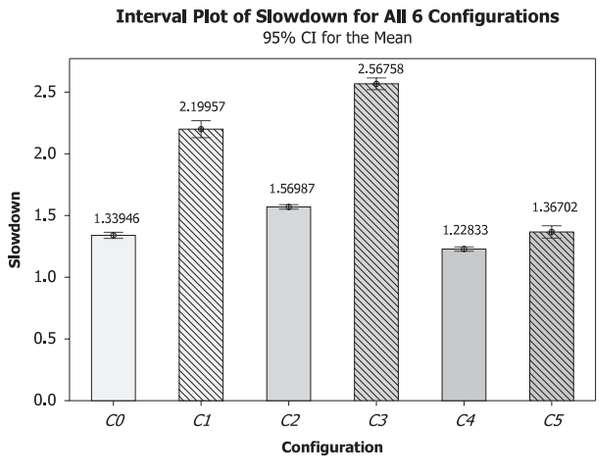


Fig. 11. Slowdown of all six configurations.

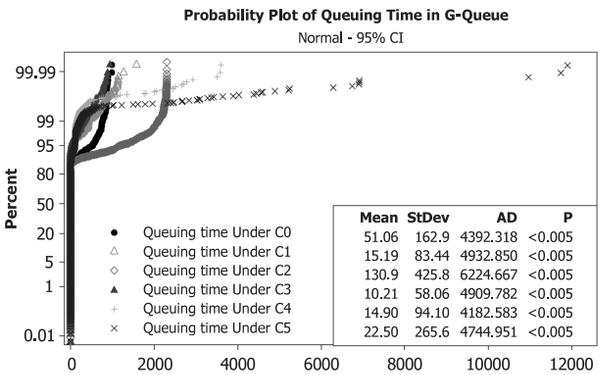


Fig. 12. Queuing time in the G-Queue.

4.4. Slowdown effects

In the emulation, we calculate the slowdown by dividing the execution time in the emulation by the execution time in the job trace. The results are presented in Fig. 11, which shows that, with the reputation mechanism, the slowdown will increase slightly, because the reputation mechanism introduces more scheduling overhead. The slowdown difference under macro-scheduling for $C4$ and $C5$ is even smaller, less than 0.04. This is because all jobs in one task have to be resubmitted once a job fails, which leads to the overwhelming workload introduced to $C4$ and $C5$. In this situation, the overhead introduced by the reputation mechanism is not distinguished any more. To this end, we argue that although the reputation mechanism introduces a slightly increased slowdown, compared to the disadvantages incurred by the job resubmission for Grid users, especially for non-IT users, they will prefer a smaller number of resubmissions even with a moderate job slowdown.

4.5. Job queuing time in the G-Queue

Another traditional metric to evaluate the scheduling performance is the job queuing time in the G-Queue. Fig. 12 illustrates the PDF of the queuing time. Under micro-scheduling ($C0$–$C3$), the mean queuing time can be reduced greatly (15.19 in $C1$ vs. 51.06 in $C0$, and 10.21 in $C3$ vs. 130.90 in $C2$). We can explain this from two angles. First for $C0$ and $C1$ without rescheduling, according to Fig. 7, all sites in $C0$ have more workload than in $C1$ except site 6 and site 8. That means that most sites in $C0$ are busier than in $C1$, which causes the waiting time in the G-Queue to be longer to get enough resources to run. Similar results can be observed in $C2$ and $C3$. Second, in $C3$ the reduction of the number of reschedulings leads to the shortening of queuing time. The queuing time in $C2$ without the reputation mechanism is much longer than (about 13 times as long as) in $C3$ with reputation. But the results are different under macro-scheduling: with reputation in $C5$ the average queuing time is 22.50, and it is 14.90 in $C4$ without reputation. The reason is that a node is considered available only when its reputation is larger than a threshold with the reputation mechanism, so some large tasks (requesting the CPU number $\geq$1000, while only site 8 and site 9 with 1024 CPUs can handle these large tasks) will find it very difficult to find sufficient number of CPUs to execute because of the existence of untrustable nodes in site 8 and 9. That will cause a considerable increase of the queuing time, and it also accounts for why $C5$ has a longer tail in Fig. 12. The maximum queuing time for these large jobs in $C5$ (around 12,000 emulation time) can be approximately 500 times the average queuing time.

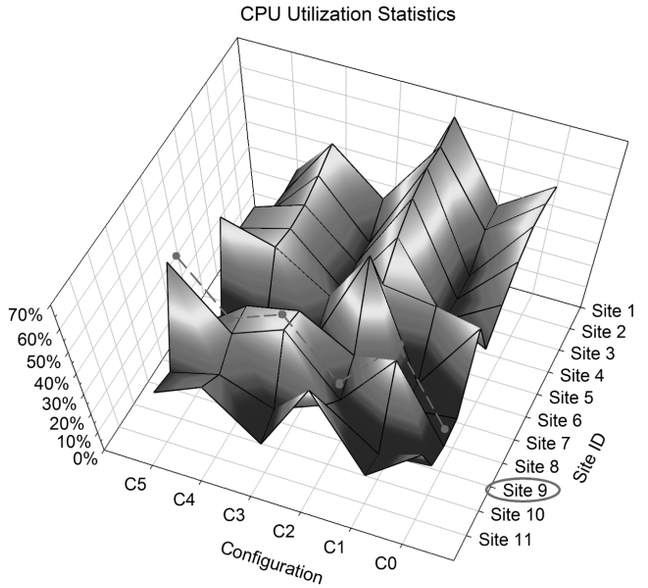


Fig. 13. CPU utilization in 11 sites under six configurations. Site 9 has a totally different utilization pattern from that of most of the other sites.

4.6. CPU utilization

Recently, saving power consumption has attracted significant attention in the Grid computing community. Most researchers prefer high resource utilization in Grid computing. But, when there are enough resources, under the presumption that there are the same number of tasks to be finished, less resource usage will be preferred, because this means that less power will be consumed. Fig. 13 shows the CPU utilization statistics for all six configurations. We can see a rough trend of the CPU utilization for all sites, that is, High $\rightarrow$ Low $\rightarrow$ High $\rightarrow$ Low $\rightarrow$ High $\rightarrow$ Low with the order of $C0$ to $C5$. This shows that, with the reputation mechanism, the CPU utilization can be reduced when there are the same number of tasks to be finished. But there are three exceptional sites, site 6, site 8 and site 9. For site 8 the reputation mechanism cannot play well, and its trend is not obvious; for site 9 where the reputation mechanism has obvious positive effects, the CPU utilization trend is exactly to the contrary: Low $\rightarrow$ High $\rightarrow$ Low $\rightarrow$ High $\rightarrow$ Low $\rightarrow$ High; site 6's pattern is close to site 9's, except for $C2$ and $C3$ where the reputation mechanism cannot completely show the potential under micro-scheduling. This exactly explains that when the reputation mechanism plays well, it will allow the good-quality (more CPUs, low job failure rate) sites to take care of most of the

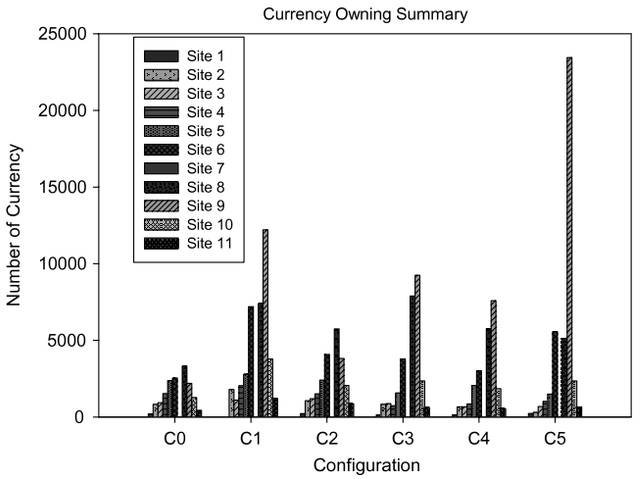


Fig. 14. Currency statistics for six configurations with 95% confidence.

workload. Users will also be content for the job to run in the good-quality sites despite the load unbalance. From Fig. 7, we can find that the total numbers of jobs run in all 11 sites are 6.32×10^5, 6.77×10^5, 7.30×10^5, 7.77×10^5, 28.97×10^5, and 11.22×10^5 from $C0$ to $C5$, respectively. So from $C0$–$C3$, where the reputation mechanism exists, the total system workload only increases a little. But for $C4$ and $C5$, using the reputation mechanism can reduce the total workload by 61.3% because its greatly improved scheduling hit leads to the job rescheduling being greatly reduced. From above, introducing the reputation mechanism is very promising in macro-scheduling by saving a lot of power with a low failure rate.

4.7. Preliminary economic effects

Although the economic model is not the focus of this paper, we still look at the preliminary economic effect for our simple economic model for the purpose of showing its promising future in the **HOURS** framework. The results are shown in Figs. 14 and 15. The results are expected to be better when a mature economic model like **M-CUBE** [31] is introduced. We hope that a site providing good service can have better income (than the sites with poor service which may be due to the limited resources or high CPU failure rate like site 7, or even with malicious intension). Fig. 14 shows the total amount of currency for each site under six configurations. We divide the results into two groups: group 1, without the reputation mechanism, including $C0$, $C2$ and $C4$; and group 2, with the reputation mechanism, including $C1$, $C3$ and $C5$. From Fig. 14, we can see that, without the reputation mechanism, the number of currencies earned by each site is basically correlated to the number of its CPUs; but for group 2, the site's income is related to its quality, which includes the number of CPUs and the job success rate. Compared to group 1, the incomes of site 8 and site 9 have a significant increase. The only exception is that site 8's income in $C5$ is less than in $C4$. In $C5$, site 8's income is even less than site 6's. This is because, although site 8 has the largest number of CPUs, site 8 has a huge number of failed jobs in $C5$ because of the unmatching of reputation and site unavailability explained in Section 4.2; it only successfully finishes 90,264 jobs, less than 130,856 in site 6, and far less than 301,284 in site 9. Fig. 15 shows the income details for sites in $C4$ and $C5$ with more details. For each (X, Y, Z) point in Fig. 15, it can be read that site X has amount of currency Z from site Y. For $C4$ without the reputation mechanism, the distribution of income of good-quality sites (site 9) is not as clear as for $C5$. In $C5$, we can clearly see that site 9's total earning is much higher than that of other sites.

However, site 9 has in total 1024 CPUs, which is the largest amount of CPUs among all sites. Intuitively, it should have more income than other sites. To better illustrate the economic and incentive problem, we introduce an economics-specific metric CPU *Value Incremental Ratio (VIR)*, which is defined as *CPU VIR = (Total Income * Job Success Rate) / CPU number*. The total income is obtained from Fig. 15, the job success rate is obtained from Fig. 8, and the CPU number is obtained from Fig. 5. Our purpose is to show that a good site (a site with large amount of CPU and high job success rate in this paper) will have more economic utility for its contributed resources with the combination of trust and economic models. A site with high VIR means its unit CPU has high revenue. Fig. 16 shows the plots of CPU VIR for all six configurations. We group them with the existence of the trust model. The result has a similar pattern as the percentage of job success rate shown in Fig. 8. For group $C0$, $C1$ and group $C4$, $C5$, the trust model can bring higher CPU VIR for almost all sites; for group $C2$, $C3$, the effects of the trust model have been reduced under the background of micro-rescheduling. We have known that site 9 is a best site from the angles of CPU number, job success rate and trustworthiness matching, from the previous discussion. In Fig. 16, we can see that, without the trust model, the CPU VIR of site 9 is not distinguished or even lower than most of other sites in Fig. 16(a) and (b); with the application of the trust model ($C1$, $C3$, $C5$), the CPU VIR of site 9 is roughly the highest amongst all sites; in particular, in Fig. 16(c), its CPU VIR is 18.64, almost two times as much as the second highest one (9.91) from site 6.

From the above analysis, we conclude that the reputation-based economic model can let the good-quality sites have more income. This is an important incentive to encourage sites to improve the

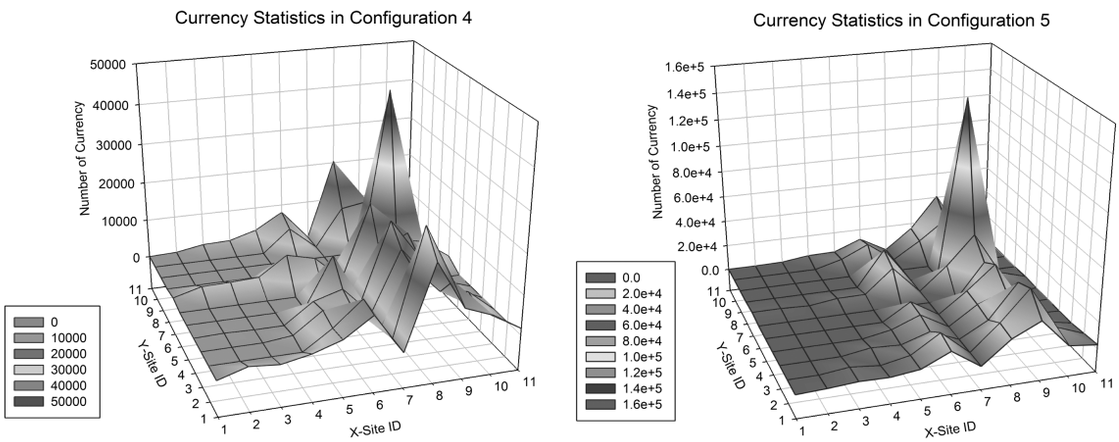
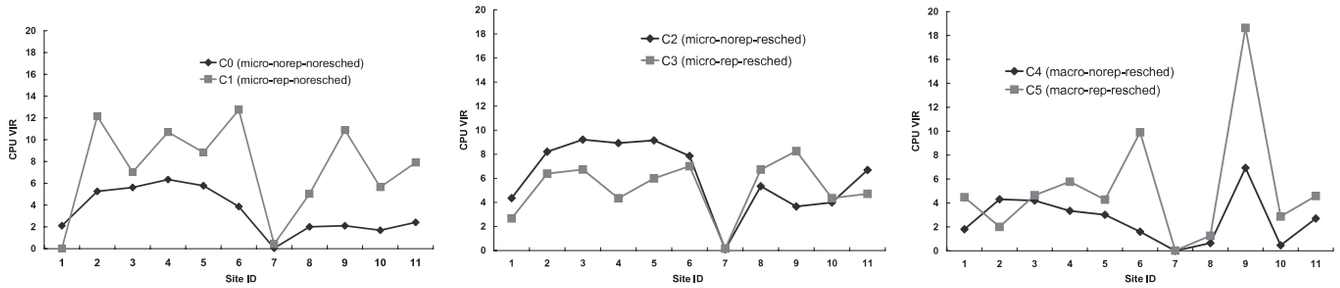


Fig. 15. Detailed currency statistics for $C4$ and $C5$.



(a) $C0$ and $C1$. (b) $C2$ and $C3$. (c) $C4$ and $C5$.

Fig. 16. CPU Value Incremental Ratio (VIR) under six configurations.

site's quality, for example, increasing the number of CPUs and maintaining the site availability. For future Grid inter-operation where different Grids contribute the resources to form a Grid of Grids, the incentive issue must be solved, and our reputation-based economic model has shown great potential. Even for the Grids supported by government, like the TeraGrid supported by NSF, introducing the reputation-based economic model can serve as an accounting incentive for maintenance purposes.

4.8. Summary

From the emulation, we observe that:

1. With the reputation mechanism, the percentage of failed jobs can be reduced, especially under $C0$, $C1$, $C4$ and $C5$. Thus **HOURS** is good for keeping a high throughput by making the main CPU time contribute to the successful jobs.

2. For large tasks, reputation-based scheduling can reduce the number of resubmissions. Under the macro-scheduling scenario, the average resubmission number for large tasks can be reduced from 3.82 to 0.70 (i.e., 543.24%).

3. Reputation-based scheduling only introduces a little more workload and increases the slowdown slightly. But the queuing time and resubmissions for macro-scheduling can be reduced significantly. Thus the introduced overhead is acceptable.

4. Reputation-based scheduling can direct the jobs to the good-quality sites. This may conflict with the goal of load balance. But for the whole system, it only introduces a little more workload. For macro-scheduling, the total CPU utilization can be reduced by 61.3% with the reputation mechanism when there are same number of tasks to be finished, which is an advantage for power saving.

5. The economic model shows the advantage of incentive introduction and accountability by leading the good-quality sites to earn more currency.

5. Related work

The notion of "trust management" was first coined by Blaze, Feigenbaum, and Lacy in their seminal paper on decentralized trust management [6]. But in the computer science literature, Marsh [36] was the first person to introduce a computational model for trust in the distributed artificial intelligence (DAI) community [36]. However, he did not model reputation in his work. After that many trust models have been proposed [7,26,32,37,42,45,50,58, 63–66]. Mui [37] gives a detailed computational model of trust and reputation. In Mui's model, reputation is well modeled, but it does not take the risk into consideration. [26,42] consider risk assessment for trust management. Different from these solutions, we make risk the assessment of the short-term behaviors and treat it as part of the trustworthiness. Different from the PowerTrust model [67] proposed by Zhou and Hwang, our trust model is an independent personalized trust model where each node has its unique local view on the trustworthiness of system, while in PowerTrust each node has one global trustworthiness value. The deficiency of PowerTrust is that it cannot reflect the real local situation or personalized experience for an independent node. It is acceptable to apply PowerTrust in a system like eBay. But for a distributed system like TeraGrid where sites/nodes may vary in having different service qualities for different sites/nodes, PowerTrust is not a good choice. For example, node A in TeraGrid can provide good quality services for node B, while bad quality services for node C. This is because nodes A and B are close and connected with a reliable high speed link, while nodes A and C may be far away, or connected with an unreliable or low speed link. This kind of service differentiation, no matter whether intentional or unintentional (due to the environment limitation), is ubiquitous in a distributed system, and so applies for TeraGrid. Our personalized trust model is able to catch the service differentiation to build more accurate personalized trust map in each independent node.

There is one major difference among these trust and reputation models; that is, these models are using different approaches of rating aggregation, i.e., how to integrate the ratings from another into one peer's own trust view. Basically, many researchers are advocating the usage of ratings and prefer complicated rating aggregation algorithms to try to filter out the bad ratings [7,20,25,45,53,60,65,66]. Wang et al. [57,58] suggest that averaging should be applied only for stranger raters, but for acquaintances, their ratings should be weighted. Yu et al. [50,64–66] give another thought on this issue. They believe that only ratings from witnesses, who have interacted with the referee (we call a peer which is recommended by raters a referee) are useful. In their weighted majority algorithm (denoted as *WMA*), only the ratings from witnesses are aggregated, and the weight of witnesses is decreased if the rating is different from its own recognition. Different from *WMA*, Sriatsa et al. [53] argue that the weight of ratings should be based on the similarity of the experience between the rater and the peer itself. We denote this approach as *personalized similarity measure (PSM)*. Finally, Jøsang et al. propose to aggregate the ratings and to update the weight of raters through deriving the expectation of the *Beta* distribution [7,20,25,60]. All these four algorithms are complicated algorithms considering the complexity of the algorithm design and the workload in the system running. Though noting the potential advantages of ratings, Resnick et al. [44] challenge the feasibility of the distribution of feedbacks, from the point of the expensive cost for the feedback distribution. Holding the same view, Liang and Shi [31,32] suggest treating the ratings from different raters equally considering the dynamics of P2P systems. They argue that simply averaging ratings is deserved considering the simplicity of the algorithm design, and the low cost in the system running. The above approaches are the major rating aggregating algorithms currently in the background of distributed trust inference.

Currently there are some projects underway or existing approaches related to the research of trust and reputation. The concept of centralized reputation systems is a very hot topic and it has been widely deployed in e-commerce [4,44,61], such as eBay (an online auction site) and slashdot.com (an online tech-guru site). Recently, in the P2P domain many decentralized reputation management schemes like P2Prep [13], EigenTrust [27], NICE project [30], and GridSec [29] have emerged. P2Prep provides a protocol complementing existing P2P protocols. Kamvar et al. [27] present EigenTrust, a distributed and secure method to compute global trust values based on "Power Iteration". Peers ask their acquaintances for their opinions about other peers to know about other peers. In [27] several threat models are described and analyzed. EigenTrust addresses these weakness by assuming there are pre-trusted nodes in the system, which is not applicable in distributed open systems. The NICE project [30] discusses trust inference problems, and [41] proposes a model to build a trustworthy software agent. The GridSec project led by Prof. Hwang [29,52] in USC is building an automated intrusion response and trust management system to facilitate authentication, authorization, and security binding in using metamorphosing Grids or P2P Web services. They propose a fuzzy reputation aggregation model to derive the trustworthiness.

Numerous economic models including microeconomics and macronomics principles for resource management have been proposed in the literature [8,48,51], and various criteria are used for judging the effectiveness of an economic model, including social welfare, stability, and computation efficiency. However, none of them takes the reputation into consideration. Several research systems have explored the use of different economic models for trading resources in different application domains: CPU cycles, storage, database query processing, and computing. Currency- and economy-based resource management has been extensively studied [19,56]. To our knowledge, the SHARP infrastructure [19] and its post work [43] is the closest work related to us, but the details of how to use the currency are different. Different from [19], our infrastructure does not favor the overbooking, which will affect its trustworthiness values at other peers. The concept of *claims* – promises or rights to control resources for designed time intervals – proposed in the SHARP system is good at coarse grain resource management, where the resource will be used for a relative long time interval. However, for fine-grain resource sharing, such as instance running of a service (e.g., serving a Web request), claim is not flexible enough as our currency model shows. PPay [62] is a micropayment-based mechanism for P2P resource sharing and it guarantees that all coin fraud is detectable, traceable and unprofitable. This work complements our work. A great deal of resource management and scheduling schemes have been proposed in the context of Grid computing, including GRAM and SNAP proposed in the context of Globus [14,15], Condor-G [18,34], GRACE [8], and Data Grid [12]. These efforts are high level and they complement the proposed economic model, which can be used to implement these high-level algorithms and policies. Iosup et al. [24] provide another economics-based approach for inter-operating Grids through delegated matchmaking, which is different from the concept of resource trading in **HOURS**.

6. Conclusion and future work

In this paper, we propose a reputation-based resource scheduler for the Grid under the background of the **HOURS** project. It is also general and flexible enough to be deployed independently in the current Grid incrementally. We are targeting to reduce the number of resubmissions and task/job failure rates. The emulation, which is a mimic of the current TeraGrid environment, shows that, using our reputation-based resource scheduling, the job failure rate can be reduced under all six configurations; under macro-scheduling, the average job resubmission number for a large task can be reduced from 3.82 to 0.70 compared to sequence resource scheduling.

The future work aims are three-fold. We will introduce multiple resource scheduling to the emulation, e.g., CPU and memory, and extend the currency model with heterogeneous resource sharing. At the same time, an advanced economic model considering more topics including advance resource reservation, SLA, pricing, and accountability will be implemented. Decided by the flexible and powerful representation capability of **HOURS**, a lot of related ongoing work can be complementary or even embedded for its future development and improvement, which includes resource reservation [49], SLA [16], resource description language [28], and automatic resource specification generation [23]. Finally, security issues are very important in the Grid, especially for the future Grid which is more open and heterogeneous. The proposed trust model and the economic model are built on some security techniques, like defense against sybil attack, DDoS attack, virus, et al. There are some existing research projects that especially focus on the security issues in the Grid [22,29,38,59]. These techniques will be integrated into the framework in the future.

Acknowledgments

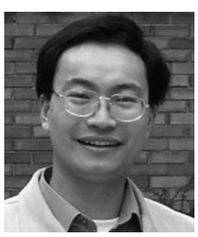
We are grateful for the constructive comments and suggestions from Dr. Jerome Lauret and Dr. Valeri Fine from Brookhaven National Laboratory, and Alexandru Iosup from Delft University of Technology. This work is in part supported by National Science Foundation CAREER grant CCF-0643521.

References

[1] La-ur-05-7318-failure-data-1996-2005.csv. http://institute.lanl.gov/data/fdata/.
[2] The San Diego supercomputer center (sdsc) datastar log. http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_ds/SDSC-DS-2004-1.swf.gz.
[3] Seti@home. http://setiathome.berkeley.edu.
[4] K. Aberer, Z. Despotovic, Managing trust in a peer-to-peer information systems, in: Proc. of the 10th International Conference on Information and Knowledge Management, CIKM'01, 2001.
[5] A. AuYoung, L. Grit, J. Wiener, J. Wilkes, Service contracts and aggregate utility functions, in: 15th IEEE International Symposium on High Performance Distributed Computing, HPDC-15, Paris, France, 2006.
[6] M. Blaze, J. Feigenbaum, J. Lacy, Decentralized trust management, in: IEEE Symposium on Security and Privacy, 1996.
[7] S. Buchegger, J.L. Boudec, A robust reputation system for p2p and mobile ad-hoc networks, in: Proc. of the Second Workshop on the Economics of Peer-to-Peer Systems, 2004.
[8] R. Buyya, D. Abramson, J. Giddy, A case for economy grid architecture for service-oriented grid computing, in: Proc. of the 10th IEEE International Heterogeneous Computing Workshop, 2001.
[9] S.J. Chapin, W. Cirne, D.G. Feitelson, J.P. Jones, S.T. Leutenegger, U. Schwiegelshohn, W. Smith, D. Talby, Benchmarks and standards for the evaluation of parallel job schedulers, in: Lect. Notes Comput. Sci., vol. 1659, 1999, pp. 67–90.
[10] L. Chen, K. Reddy, G. Agrawal, Gates: A grid-based middleware for processing distributed data streams, in: HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society, Washington, DC, USA, 2004, pp. 192–201.
[11] L. Chen, Q. Zhu, G. Agrawal, Supporting dynamic migration in tightly coupled grid applications, in: SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, ACM, New York, NY, USA, 2006, p. 117.
[12] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets, Journal of Network and Computer Applications 23 (3) (2000).
[13] F. Cornelli, E. Damiani, S.D.C. Vimercati, S. Paraboschi, P. Samarati, Choosing reputable servents in a p2p network, in: Proc. of the 11th International World Wide Web Conference, 2002.
[14] K. Czajkowski, I. Foster, C. Kesselman, Resource co-allocation in computational grids, in: Proc. of Eighth International Symposium on High Performance Distributed Computing, HPDC-8, 1999.
[15] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, S. Tuecke, Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems, in: Lecture Notes in Computer Science, vol. 2537, 2002.

[16] C. Dumitrescu, I. Raicu, I. Foster, Di-gruber: A distributed approach to grid resource brokering, in: SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, 2005, p. 38.
[17] EGEE. http://public.eu-egee.org/.
[18] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-g: A computation management agent for multi-institutional grids, in: Proc. of Tenth International Symposium on High Performance Distributed Computing, HPDC-10, 2001.
[19] Y. Fu, J. Chase, B. Chun, S. Schwab, A. Vahdat, Sharp: An architecture for secure resource peering, in: Proc. of the 19th ACM Symp. on Operating Systems Principles, SOSP-19, 2003.
[20] S. Ganeriwal, M. Srivastava, Reputation-based framework for high integrity sensor networks, in: Proc. of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks, Washington DC, USA, 2004.
[21] The Globus Project. http://www.globus.org.
[22] GridSec. http://gridsec.usc.edu/.
[23] R. Huang, A. Chien, H. Casanova, Automatic resource specification generation for resource selection, in: ACM/IEEE Conference on High Performance Networking and Computing, SuperComputing 2007, 2007.
[24] A. Iosup, D. Epema, T. Tannenbaum, M. Farallee, M. Livny, Inter-operating grids through delegated matchmaking, in: ACM/IEEE Conference on High Performance Networking and Computing, SuperComputing 2007, 2007.
[25] A. Jøsang, R. Ismail, The beta reputation system, in: Proc. of the 15th Bled Electronic Commerce Conference, 2002.
[26] A. Jøsang, S.L. Presti, Analysing the relationship between risk and trust, in: Proc. of the Second International Conference on Trust Management, 2004.
[27] S. Kamvar, M.T. Schlosser, H. Garcia-Molina, The eigentrust algorithm for reputation management in p2p networks, in: Proc. of the 12th International World Wide Web Conference, 2003.
[28] Y. Kee, K. Yocum, A.A. Chien, H. Casanova, Improving grid resource allocation via integrated selection and binding, in: SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, 2006, p. 99.
[29] K. Hwang, Y. Kwok, S. Song, et al. Gridsec: Trusted grid computing with security binding and self-defense against network worms and ddos attacks, in: The International Workshop on Grid Computing Security and Resource Management, GSRM'05, 2005.
[30] S. Lee, R. Sherwood, B. Bhattacharjee, Cooperative peer groups in nice, in: Proc. of IEEE Conference on Computer Communications, INFOCOM'03, 2003.
[31] Z. Liang, W. Shi, Enforcing cooperative resource sharing in untrusted peer-to-peer environment, ACM Journal of Mobile Networks and Applications (MONET) 10 (6) (2005) 771–783 (special issue on Non-cooperative wireless networking and computing).
[32] Z. Liang, W. Shi, PET: A PErsonalized Trust model with reputation and risk evaluation for P2P resource sharing, in: Proc. of HICSS-38, Hilton Waikoloa Village Big Island, Hawaii, 2005.
[33] Z. Liang, W. Shi, Analysis of recommendations on trust inference in open environment, Journal of Performance Evaluation 65 (2) (2008) 99–128.
[34] M. Litzkow, M. Livny, M. Mutka, Condor — A hunter of idle workstations, in: Proc. of the 8th International Conference of Distributed Computing Systems, 1988, pp. 104–111.
[35] M. Litzkow, M. Solomon, Supporting checkpointing and process migration outside the Unix kernel, 1999, pp. 154–162.
[36] S. Marsh, Formalising trust as a computational concept, Ph.D. Thesis, University of Stirling, 1994.
[37] L. Mui, Computational models of trust and reputation: Agents, evolutionary games, and social networks, Ph.D. Thesis, Massachusetts Institute of Technology, 2002.
[38] OGSA-SEC-WG Draft, Security architecture for open grid services, Jun 2003. http://www.globus.org/toolkit/security/ogsa/draft-ggf-ogsa-sec-arch-01.pdf.
[39] Open Science Grid. http://www.opensciencegrid.org.
[40] M. Parashar, S. Hariri, Autonomic computing: An overview, UPP 3566 (2004) 247–259.
[41] A.S. Patrick, Building trustworthy software agents, IEEE Internet Computing (2002) 46–53.
[42] A.A. Rahman, S. Hailes, Supporting trust in virtual communities, in: Proc. of 33rd Hawaii International Conference on System Sciences, Maui, Hawaii, 2000.
[43] L. Ramakrishnan, D. Irwin, L. Grit, A. Yumerefendi, A. Iamnitchi, J. Chase, Toward a doctrine of containment: Grid hosting with adaptive resource control, in: SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, 2006, p. 101.
[44] P. Resnick, R. Zeckhauser, E. Friedman, K. Kuwabara, Reputation systems, Communications of the ACM 43 (12) (2001) 45–48.
[45] J. Sabater, C. Sierra, Regret: Reputation in gregarious societies, ACM SIGecom Exchanges 3 (2002).
[46] B. Schroeder, G.A. Gibson, A large-scale study of failures in high-performance computing systems, in: DSN '06: Proceedings of the International Conference on Dependable Systems and Networks, IEEE Computer Society, Washington, DC, USA, 2006, pp. 249–258.
[47] B. Schroeder, G.A. Gibson, Understanding failures in petascale computers, Journal of Physics: Conference Services 78 (2007).
[48] J. Shneidman, C. Ng, D.C. Parkes, A. AuYoung, A.C. Snoeren, A. Vahdat, B.N. Chun, Why markets could (but don't currently) solve resource allocation problems in systems, in: Proc. of the 10th USENIX Workshop on Hot Topics in Operating Systems, HotOS-X, Santa Fe, NM, 2005.
[49] M. Siddiqui, A. Villazón, T. Fahringer, Grid capacity planning with negotiation-based advance reservation for optimized qos, in: SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, 2006, p. 103.
[50] M.P. Singh, Trustworthy service composition: Challenges and research questions, in: Proc. of the 1st International Joint Conference on Autonomous Agents and MultiAgent System, AAMAS, 2002.
[51] R. Smith, R. Davis, The contract net protocol: High level communication and control in a distributed problem solver, IEEE Transactions on Computers C-29 (12) (1980) 1104–1113.
[52] S. Song, K. Hwang, Trusted grid computing with security assurance and resource optimization, in: 17th International Conference on Parallel and Distributed Computing Systems, PDCS-2004, 2004.
[53] M. Srivatsa, L. Xiong, L. Liu, Trustguard: Countering vulnerabilities in reputation management for decentralized overlay networks, in: Proc. of 14th World Wide Web Conference, WWW, 2005 (in press).
[54] TeraGrid. http://www.teragrid.org.
[55] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: The condor experience, Concurrency—Practice and Experience 17 (2–4) (2005) 323–356.
[56] C.A. Waldspurger, W.E. Weihl, Lottery scheduling-flexible proportional-share resource management, in: Proc. of the First Symposium on Operating Systems Design and Implementation, Usenix Association, 1994.
[57] Y. Wang, J. Vassileva, Bayesian network-based trust model, in: Proc. of IEEE/WIC International Conference on Web Intelligence, WI 2003, Halifax, Canada, 2003.
[58] Y. Wang, J. Vassileva, Trust and reputation model in peer-to-peer networks, in: Proc. of Third International Conference on Peer-to-Peer Computing, P2P'03, 2003.
[59] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke, Security for grid services, in: HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society, Washington, DC, USA, 2003, p. 48.
[60] A. Whitby, A. Jøsang, J. Indulska, Filtering out unfair ratings in Bayesian reputation systems, in: Accepted for the Autonomous Agents and Multi Agent Systems 2004, AAMAS-04, Workshop on "Trust in Agent Societies", New York, 2004.
[61] L. Xiong, L. Liu, A reputation-based trust model for peer-to-peer ecommerce communities, in: Proc. of the IEEE Conference on E-Commerce, 2003.
[62] B. Yang, H. Garcia-Molina, Ppay: Micropayments for peer-to-peer systems, in: Proc. of ACM CCS'03, 2003.
[63] P. Yolum, M.P. Singh, Emergent properties of referral systems, in: Proc. on Autonomous Agents and Multiagent Systems, 2003.
[64] B. Yu, M.P. Singh, A social mechanism of reputation management in electronic communities, in: Proc. of Fourth International Workshop on Cooperative Information Agents, Berlin, 2000.
[65] B. Yu, M.P. Singh, Searching social networks, in: Proc. of the 2nd International Joint Conference on Autonomous Agents and MultiAgent System, AAMAS, Melbourne, 2003.
[66] B. Yu, M.P. Singh, K. Sycara, Developing trust in large-scale peer-to-peer systems, in: Proc. of First IEEE Symposium on Multi-Agent Security and Survivability, 2004.
[67] R. Zhou, K. Hwang, Trusted overlay networks for global reputation aggregation in p2p grid computing, in: 20th IEEE International Parallel and Distributed Processing Symposium, IPDPS'06, Greece, 2006.

Zhengqiang Liang is a Ph.D. candidate in computer science at Wayne State University. His researches focus on trusted and cooperative resource sharing in the open environment, trust based resource scheduling in open scientific discovery infrastructure, next generation internet, P2P systems, and computer economics. He received his B.S. degree in 2001 and M.S. degree in 2003 from Harbin Institute of Technology (HIT) in China, both in Computer Science and Engineering.



Weisong Shi is an Associate Professor of Computer Science at Wayne State University. He received his B.S. from Xidian University in 1995, and Ph.D. degree from the Chinese Academy of Sciences in 2000, both in Computer Engineering. His current research focuses on mobile computing, distributed systems and high performance computing. Dr. Shi has published more than 80 peer-reviewed journal and conference papers in these areas. He is the author of the book "Performance Optimization of Software Distributed Shared Memory Systems" (High Education Press, 2004). He has also served on technical program committees of several international conferences, including WWW, ICPP, and MASS. He is a recipient of a Microsoft Fellowship in 1999, the President outstanding award of the Chinese Academy of Sciences in 2000, one of 100 outstanding Ph.D. dissertations (China) in 2002, "Faculty Research Award" of Wayne State University in 2004 and 2005, Career Development Chair Award of Wayne State University in 2009, and the "Best Paper Award" of ICWE'04 and IPDPS'05. He is a recipient of the NSF CAREER award.