A General Self-adaptive Task Scheduling System for Non-dedicated Heterogeneous Computing

Ming Wu, *Student Member, IEEE,* Xian-He Sun, *Senior Member, IEEE*

***Abstract*— The efforts to construct a national scale Grid computing environment have brought unprecedented computing capacity and complicacy. Exploiting this complex infrastructure requires efficient middleware to support the execution of distributed applications, which presents the challenge on how to schedule tasks in shared heterogeneous systems. Most existing scheduling systems are based on pre-determined estimation of task completion time and resources availability. They may not provide appropriate scheduling if the underlying computing resources present an abnormal usage pattern during an application execution. For solving long-running applications in a large-scale Grid environment, abnormal usage of some resource may not be uncommon. We have proposed the development of the Grid Harvest Service (GHS) performance evaluation and task scheduling system in our previous work. In this study, we present a novel dynamic self-adaptive scheduling algorithm and its implementation under GHS. Scheduling and re-scheduling algorithms and mechanisms are carefully investigated. Experimental results show that, equipped with these new scheduling mechanisms, GHS outperforms existing systems considerably in scheduling large applications in a non-dedicated heterogeneous environment.**

***Index Terms*— task scheduling, performance prediction, shared heterogeneous system.**

I. Introduction

In order to provide high performance computation power to serve the increasing need of large applications, people strive to improve a single machine's capacity or construct a distributed system composed of a scalable set of machines. Compared to the former, where the improvement is mainly up to the hardware technology development, the construction of distributed systems for resource collaboration is more complex. Some of well-known existing distributed systems composed of heterogeneous resources are Condor [1], NetSolve [2], Nimrod [3], Globus, and the Grid computation environment [4]. These systems, especially the Grid, have unprecedented computing power. However delivering this unprecedented computing power to the users is still an elusive problem. One of the major issues is how to schedule a large application in these non-dedicated distributed systems [5]. In general, scheduling applications in a distributed system is a NP-hard problem [6]. Many heuristic scheduling algorithms and systems are proposed to address this problem. Unfortunately, most of scheduling algorithms proposed so far are for dedicated systems. By dedicated, we mean the resources are dedicated for a given application. In contrary most current distributed systems are non-dedicated, shared environments [7].

Good scheduling in a shared environment involves the integration of application specific information and system specific information [8]. We studied a performance-prediction based task scheduling system, which provides task allocation and scheduling based on application-level and system-level performance prediction. The effects of system specific information, such as utilization, job service rate, job arrival rate, and application-specific information, such as workload, divisibility, parallel processing, on the application performance have been identified. The preliminary results demonstrate the effectiveness of our scheduling mechanism for long-term applications [9]. Our earlier work focuses on the task scheduling for parallel processing, which assumes that the total workload of a parallel application could be arbitrarily partitioned. In this study, we extend our scheduling policy for a class of widely used Grid application, the parameter-sweep application. It is composed of a set of independent and indivisible tasks [10].

A key question is that how to maintain an application performance during its execution? When some of resources assigned for subtasks of the application represent abnormal status from their history information, the completion times of the subtasks are different from the estimation. Thus the whole application performance will change under the situation of resource abnormality. There is a fair chance that a resource may show different usage patterns from its history in a large distributed system. In this paper, we only consider resources are overloaded when we mention resource abnormality. If the application is a long-running job (days or weeks), the

Ming Wu is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 USA (e-mail: wuming@iit.edu).

Xian-He Sun is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 USA (phone: 312-567-5260; fax: 312-567-5067; e-mail: sun@cs.iit.edu).

performance degradation caused by resource abnormality is unacceptable. To provide a robust task scheduling system working in a dynamic system, we introduce a self-adaptive task scheduling algorithm, which monitors the long-running application progress and detect possible resource abnormality. The self-adaptive scheduling algorithm selects appropriate resources and reassigns the subtasks on abnormal machines to these selected machines based on the application-level prediction.

Our goal in task scheduling system is to provide a robust and general-purpose scheduling system. The task scheduling is divided into three parts, allocator, predictor and scheduler. Allocator selects appropriate allocation algorithms to partition or group the subtasks of a divisible application on a number of machines, where a divisible application refers to an application that could be partitioned into or composed of a set of subtasks [11]. Predictor estimates the application execution time distribution on each machine. Scheduler decides which set of machines is the best among all possible sets of machines. In this way, we provide a flexible task scheduling mechanism for the need of different application-level scheduling scenarios. The proposed scheduling system is very flexible: on one hand it could be easily integrated with other scheduling systems, while on the other hand other people's work in different aspects of task scheduling can be easily incorporated into our work.

The rest of this paper is organized as follows: Section 2 describes the related work. Section 3 introduces our proposed task-scheduling system design. Various scenarios of task allocation and scheduling are discussed in this section. A meta-task scheduling algorithm is presented and a self-adaptive task scheduling is introduced with three performance metrics. Experimental results are presented in Section 4. We compare GHS scheduling system with a well-known scheduling system, AppLeS, and verify the efficiency of the self-adaptive scheduling algorithm. Finally we conclude and summarize our work with section 5.

II. Related Work

Related earlier work in task scheduling policy mainly focused on dedicated systems, which cannot be assumed in non-dedicated computing environment like the Grid. A resource reservation strategy [12] is proposed to address this problem. Shared resources are reserved in advance for a user's dedicated occupancy. Reservation is a good way to conduct experimental testing under current environments. However, the utilization of a reservation system in general is low. It is quite often a user has to wait to a few days to get his/her required resources in current NCSA grid environments. This is not feasible in a production environment. In addition, private owners usually don't want their machines reserved for others and like to access their machines when they need. Task scheduling on such a shared non-dedicated computing environment offers a big challenge. A task scheduling mechanism based on the prediction of machine availability has been investigated in [8]. This determined method might be simple and good as it is useful for scheduling short-running applications in a single machine or a small system. However, due to the internal limitation of its determined prediction method, it is not suitable for scheduling a long-running application in a large distributed system since the prediction error caused by the fluctuation of resource status increases with the number of resources running applications. Yan Alexander Li and John K. Antonio developed a probabilistic approach to estimate the execution time of a parallel application in a heterogeneous computing system. However, their model is based on the assumption that the execution time distribution of an individual task on a machine is pre-known [13]. And the effect of system specific information on the application performance is not reflected in their work.

Different task scheduling policies are applied in various heterogeneous computing environments. GASA (Grid Advance Reservation API) [12] is a subsystem of Globus project. It provides mechanisms for resource reservation so that a remote application can receive a certain level of service from a resource. As we mentioned before, this policy doesn't favor the privilege of the private owner of a shared resource. Condor system [1] provides a matchmaking mechanism to allocate resources with ClassAds. The scheduling strategy is based on the mapping of the users' ClassAds, which specify the job requirements and users' preferences, and the machines' ClassAds, which advertise their characteristics, available time period, and conditions. Instead of the performance issue, the economy issue is considered as the evaluation criteria in Nimrod project. Legion system [14] also supports resource reservation. It focuses on providing basic mechanisms for building application-level scheduling algorithms rather than constructing scheduling algorithm itself. A simple random selection policy is provided as the default scheduling mechanism. Currently, scheduling algorithms in the AppLeS [10], [15] project are supported by the short-term system prediction provided by NWS services [16], [17]. Although they did consider resource availability in making the scheduling decision, none of them fully identifies the effect of the "shared" characteristic of resources on the execution time of an application. Instead of submitting all subtasks of an application at once to the system, people use a loop of task events to schedule task dynamically. However, it is hard to define what is enough work for one schedule event. They don't consider rescheduling during the application execution when some resources show abnormality. Their system works well under the assumption that the deterioration of subtask performance would not affect the whole application performance, which is not a general case in distributed computing.

III. Task Scheduling System

Prediction of application and system performance is

necessary for a good scheduling [8]. Some of current scheduling systems do involve some prediction work. However due to the determined approach of their simple performance models, the effect of various system parameters on the application performance is not available in their systems. The performance prediction in the proposed scheduling system is based on probabilistic modeling. The effect of system specific information on application performance has been identified by our general performance model [18]. The application performance prediction is provided by the application-level predictor, which is one of the major components of our scheduling system. A meta-task scheduling algorithm was proposed to address the problem of scheduling a set of independent subtasks in a distributed environment. A new task allocation method, a min-min task allocation, has been developed to support meta-task scheduling. To reduce the performance loss brought by abnormal resource usage pattern, a self-adaptive scheduling algorithm has been investigated. We studied the capability of the self-adaptive scheduling algorithm identifying abnormal machines and the efficiency of the self-adaptive algorithm in term of performance loss reduction rate. We focus on scheduling computation-intensive applications.

A. System architecture

A block diagram of GHS scheduling system design is shown in Fig. 1. A user submits an application with its characteristics (application type, workload) to the Task Manager. It inquires the Task Scheduler for qualified scheduling solution. By accessing the resource information provided by the Resource Information Service, the Task Scheduler finds a list of potential resources. The Task Scheduler searches possible task allocation plans. The Task Allocator decides how to map subtasks of an application among resources based on the prediction of system status provided by the System-level predictor. The map of subtasks on machines will be forwarded to the Application-level Predictor to estimate the application performance. The best scheduling solution satisfying the evaluation criteria is returned to the Task Manager. When the user is satisfied with the expected application performance, the application will be submitted for running in the distributed computation environment through the Task Execution Service. The Task Manager monitors the application execution and may invocate the Task Scheduler to reschedule the application in the system. The resource information is collected through various Sensors and stored in the Resource Information Service, which provides the access of resource information for the System-level Predictor and the Task Scheduler.

B. Prediction

The goal of application-level predictor is to estimate the application execution time in a shared distributed system. The difference between our application-level predictor and existing scheduling systems [19], [20] is that we estimate the application make-span based on probabilities analysis instead of determined approaches used by other scheduling systems.

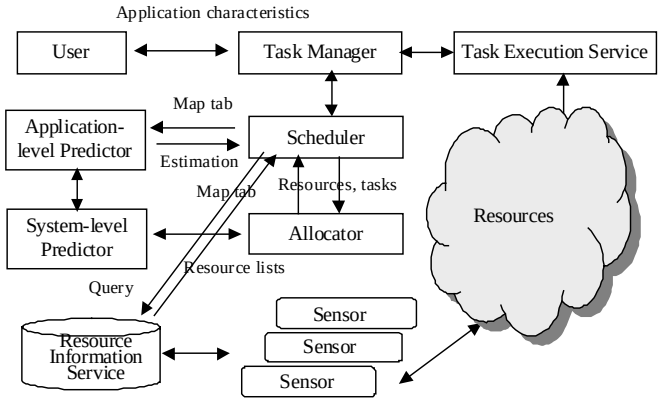

Fig. 1. A framework of GHS scheduling system.

So the effect of both system specific information and application specific information on the application performance are identified in our application-level prediction. Based on the observation of machine usage pattern [21]-[23], we assume that the local job processing follows M/G/1 queuing system. The system parameters, λ, ρ and σ, reflect the system characteristics of a shared resource in a non-dedicated distributed system. ρ is the system utilization. λ is the arrival rate of local jobs and σ is the standard deviation of service time of local jobs. The workload and divisibility of an application reflect the general characteristics of the application. If the application is a single indivisible task, the cumulative distribution function of the application completion time on a machine can be calculated as [9]:

$$\Pr(T \le t) = \begin{cases} e^{-\lambda w/\tau} + (1 - e^{-\lambda w/\tau}) \Pr(U(S) \le t - w/\tau \mid S > 0), & \text{if } t \ge w \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where $U(S)$ is the sum of busy periods of local jobs on the machine. We denote τ the computing capacity of the machine and w the workload of the application. If the workload of an application is divisible, we can use $\Pr(T \le t) = \prod \Pr(T_k \le t)$ to calculate the cumulative distribution function of the application completion time (T) where T_k is the subtask completion time on machine M_k. The sub-workload on each machine, w_k, is calculated by the Task allocator component.

C. Task allocation

When an application is a single indivisible task, we choose a machine where the sum of the expectation and variance of the application execution time is minimal. In general, however, an application in a distributed system is likely to be solved concurrently for best performance. This would happen in two scenarios. The first is that the application can be partitioned into subtasks arbitrarily. The other scenario is that the application consists of independent subtasks. These subtasks cannot be partitioned further. In this scenario, there are two more situations are worthy of concern. The first one is that the subtasks have no dependency among each other. We define such an application as a meta-task. Other situation is that there are dependencies among subtasks. In this paper, we study

meta-task scheduling. An ideal example of such applications is the class of parameter-sweep applications, which are composed of a set of independent and indivisible tasks [10].

For a given set of resources, the Task Allocator decides how to partition/group subtasks of an application. If the workload of an application is divisible, the task allocation is a partition problem. If an application is a meta-task, the task allocation becomes a task grouping issue. We have developed two allocation algorithms respectively for the two classes of applications. In the former situation, a mean-time allocation algorithm is applied. The workload is assigned to each resource so that the expected mean execution time of subtasks on each resource is the same.

When an application is composed of independent indivisible subtasks, the Task Allocator group the subtasks and map each subtask group to one of resources for optimal performance based on the estimation of subtask execution time. Assume that we have a meta-task composed of a number of independent tasks, $\{T_1, T_2, ..., T_p\}$ with workload $\{w_1, w_2, ..., w_p\}$, a list of machines $\{M_1, M_2, ..., M_q\}$, and a task allocation solution $\{TG_1, TG_2, \cdots TG_q\}$ where TG_k is the set of subtasks on machine M_k. the cumulative distribution function of the application completion time on a machine is

$$\Pr(T \le t) = \prod_k \begin{cases} e^{-\lambda w_k/\tau} + (1-e^{-\lambda w_k/\tau}) \Pr(U(S) \le t - w_k/\tau \mid S > 0), & \text{if } t \ge w_k \\ 0, & \text{otherwise} \end{cases}$$

where w_k = {the sum of workloads of all subtasks on machine M_k}.

Our goal is to find a task allocation solution so that the mean completion time of the meta-task is the minimum. This is a NP-hard problem. Heuristic methods are applied. We propose a min-min task group allocation algorithm. The basic idea is similar to the mean-time allocation algorithm. The subtasks of a meta-task is first grouped and then each subtask group is mapped to resources so that the difference of the expected execution time of subtasks on each resource is the minimum. Fig. 2 shows the min-min task group allocation algorithm in details.

Assumption: a meta-task composed of a number of independent subtasks, $\{T_1, T_2, ..., T_p\}$ and a list of machines $\{M_1, M_2, ..., M_q\}$. Each subtask is indivisible.

Objective: find a task group $TG_k = \{T_{k_1}, T_{k_2}, ..., T_{k_q}\}$ for machine M_k $(1 \le k \le q)$.

Begin

/* W_k means the current workload on machine M_k */

$W_k = 0$; $TG_k = \phi$; $(1 \le k \le q)$; $i = 1$;

While $i < p$

  $j = 1$;

  While $j < q$

    $E(T_{i,j}) = w_i / [\tau_j * (1 - \rho_j)]$; $j = j + 1$;

  End While

  Choose machine $M_{k'}$ where

    $W_{k'} / [\tau_{k'} * (1 - \rho_{k'})] + E(T_{i,k'})$ is minimal.

  $TG_{k'} = TG_{k'} \cup \{T_i\}$,

  $W_{k'} = W_{k'} + w_i$; $i = i + 1$;

End While

Return TG_k and W_k $(1 \le k \le q)$;

End

Fig. 2. Min-min task group allocation algorithm.

D. Scheduling algorithms

1) Meta-task scheduling: The Task Scheduler is responsible for finding a potential machine set for users' applications in a heterogeneous environment. It is supported by the Task Allocator and the Application-Level Predictor. The Task Scheduler supports different scheduling scenarios according to the application characteristics. Our earlier work studied single sequential task scheduling and optimal scheduling for parallel program. If the application can be partitioned into any number of subtasks, we have to search 2^n possible degree of parallelisms and machine combinations. To reduce the scheduling cost when the machine set is large, a corresponding heuristic task-scheduling algorithm was therefore proposed to find an acceptable solution with a reasonable cost. In this study, we extend our earlier scheduling algorithm for a meta-task, which is composed of independent subtasks. A good example of meta-task is the parameter sweep application, a widely used grid application.

The parameter-sweep application is composed of a set of subtasks. Each subtask is independent of each other and is indivisible. A subtask reads data from input files, computes data, and generates an output file. This process doesn't need to coordinate with other subtasks. The relationship between subtasks is that some of them may share a same input file. AppLeS system has been used to schedule the parameter sweep application in Grid computation environment. However it is based on the estimation of the mean execution time of subtask on a single machine. Our scheduling decision is based on the estimation of the performance of the whole parameter sweep application on a set of machines. Fig. 3. gives the optimal meta-task scheduling algorithm for parameter-sweep applications. Please notice a similar heuristic method could be applied in searching possible scheduling plans to reduce the cost of scheduling in a large-scale system.

In this study, we focus on scheduling computation-intensive applications. Thus we assume that the communication cost of transferring initial input files to each site is relatively small compared with the computation time of a cluster of subtasks on each site. We utilize NWS to predict the file transferring time in the application execution initialization because NWS supports the prediction of the data communication time up to 5 minutes. The estimate of the communication time for file transfers provided by NWS could be easily integrated into the scheduling algorithm in Fig. 3 to predict the completion time of a meta-task. The similar method has been applied in AppLeS. Interested readers can refer to [10] for details.

2) Self-adaptive scheduling: In the grid computation environment, most of resources are shared. There is a fair chance that some of resource may represent abnormal performance from their historical records, especially when the size of machine set is large. If the performance of some resources running subtasks of the scheduled application is degraded, the completion time of the application will increase. How to decide whether we should reschedule the application and which resource we should choose are the problems of consideration. A self-adaptive task scheduling algorithm is thus proposed to address this issue. The basic idea is to assign subtasks on a resource showing abnormal performance to other appropriate resources so that the application performance degradation could be reduced. The subtasks could be either migrated or restarted on the selected machine, which is up to the migration cost and whether a migration is possible. The selection of machines is decided based on the estimation of the whole application completion time on the updated machine set. The criterion of introducing a new scheduling plan is whether the new plan brings benefit for the application performance. Because there is no general performance model to estimate the migration cost and the communication cost of process migration may be overlapped with the computation of subtasks, we ignore the effect of the migration cost on the rescheduling decision at this time. When the user can estimate that cost, the cost can be added into our self-adaptive scheduling directly.

The estimation of application performance is based on the measurement of system parameters. The details of measuring mechanism have been given in our previous work [9]. We use the same techniques to measure λ_k, σ_k and τ_k because the running of the scheduled application doesn't affect these system parameters. However, for system utilization, ρ_k, we can not use *vmstat* utility to measure it since the resource utilization is theoretically close to 100% during the application execution. We use the *ps* utility to measure the subtask CPU time. The ratio of the subtask CPU time and its real system time will be used as the estimate of ρ_k.

To decide whether the rescheduling plan benefit the application performance, we need to calculate the application execution time in both situations: with and without rescheduling. An extension of (1) is used to calculate the cumulative distribution function of the application execution time in both situations:

$$\Pr(T \le t) = \begin{cases} (\prod_{i=1}^{m} \Pr(T_i \le t), & if \quad t \ge w_{\max} \\ 0, & otherwise \end{cases} \qquad (2)$$

where

$$\Pr(T_i \le t) = e^{-\lambda_i w_i' / \tau_i} + (1 - e^{-\lambda_i w_i' / \tau_i}) \Pr(U(S_i) \le t - t_i - w_i' / \tau_i \mid S_i > 0).$$

t_i denotes the execution time of subtasks on machine i so far and w_i' denotes the workload of subtasks on machine i that haven't been completed. $w_{\max} = \max mum\{t_i + w_i'\}$.

During the execution of the application, we check running information of subtasks on each machine periodically. We compare the difference between the expected ρ_k and the real ρ_k. If the difference is more than a threshold, we assume the resource suffers abnormality. The set of the monitor period is related to the set of the threshold. In the next section, we conducted experiments with different monitor periods and thresholds. Using (2), we calculate the mean of the completion time of the scheduled application with and without reassignment in the new situation. We denote the former as $E(T_{reassign})$ and the latter $E(T_{original})$. If $E(T_{original}) - E(T_{reassign}) > 0$, that indicates move/migration will benefit the application performance. We select resources that can bring the most benefit if we assign subtasks on an abnormal resource to them. A detailed description of the self-adaptive scheduling algorithm is given in Fig. 4. In this algorithm, we assume that there are two situations of task reallocation. The first is one-to-one task reallocation, which means that all uncompleted subtasks on an abnormal machine will be moved as a unit to another machine if the reallocation can benefit the application running time. The second is one-to-all task reallocation, which means that the uncompleted subtasks on an abnormal machine could be moved separately to any possible machine to favor application make-span. In this situation, we invoke the min-min task group allocation algorithm discussed in Section 4.3 to map the uncompleted subtasks on an abnormal machine into other machines. We set

Assumption: a meta-task is composed of a number of independent subtasks. These subtasks are indivisible.
Objective: Scheduling a meta-task

Begin
List a set of idle machines that are lightly loaded over an observed time period, $M = \{M_1, M_2, ..., M_q\}$;
$p' = 1$; $k' = 1$;
$p = 1$;
While $p < q$
  List all the possible sets of machines, $S^p = \{S_1^p, S_2^p, \ldots S_z^p\}$, $S_i^p \subset M$ and $|S_i^p| = p$;
  For each machine set S_k^p $(1 \le k \le z)$,
    Use task group allocation algorithm to group subtasks to each machine in S_k^p;
    Calculate $E(T_{S_k^p})(1 + Coe.(T_{S_k^p}))$;
    If $E(T_{S_{k'}^{p'}})(1 + Coe.(T_{S_{k'}^{p'}})) > E(T_{S_k^p})(1 + Coe.(T_{S_k^p}))$,
      Then $p' = p$; $k' = k$;
    End If
  End For
  $p = p + 1$;
End While
Assign the meta-task to the machine set $S_{k'}^{p'}$;
End

Fig. 3. Optimal meta-task scheduling algorithm.

$W_k = 0$, $TG_k = \phi$ for the identified abnormal machines and $W_k = \{$the sum of workload of completed subtasks$\}$ and $TG_k = \{$the set of completed subtasks$\}$ for other machines. The cost of the self-adaptive scheduling algorithm is mainly decided by the cost of the search of an appropriate machine set for task reallocation. We have proposed a heuristic scheduling algorithm to search a near-optimal solution [9]. The earlier experiment data demonstrated its cost is ignorable.

3) Performance metrics: We have discussed how to reschedule when we find resources showing irregular status. To evaluate the performance of the self-adaptive scheduling algorithms, we have to answer the following questions:

a. Whether the algorithm can identify the abnormal machine? What is the error rate?

b. Whether the self-adaptive scheduling algorithm can reduce the application performance loss due to resource abnormality?

c. What is the appropriate utilization threshold to benefit the application performance?

The key to a successful reschedule is that the self-adaptive task scheduling algorithm can rightly identify the abnormal machines where the performance degradation happens. The difficulty in identifying abnormal machines in a shared environment is that resource usage is dynamically changing. It is natural that a normal machine may present a high-level resource utilization during some periods and in the other hand, an abnormal machine may present a low-level resource utilization sometimes. Thus there is a possibility that we may identify an abnormal machine as a normal machine or a normal machine as an abnormal machine sometimes. Both situations will lead to a false rescheduling plan and thus an application performance deterioration. This is inevitable in a shared environment. Our aim is to find an appropriate utilization threshold so the wrong identification rate could be kept in a low level.

A perfect identification of abnormal machines means that we successfully identify all abnormal machines while we don't make any mistake to take any normal machine as an abnormal machine. We use two metrics to describe the capability of the self-adaptive scheduling algorithm identifying abnormal machines, right identification of abnormal machine rate (RIAM) and right identification of normal machine rate (RINM). RIAM denotes the ratio of the number of rightly identified abnormal machines to the number of actual abnormal machines. RINM is defined as $1 - N_f / N_n$ where N_f denotes the number of identified abnormal machines that are actually normal machines and N_n denotes the number of normal machines. N_f / N_n could be viewed as the error rate of identification of normal machine. To evaluate the efficiency of the self-adaptive algorithm in reducing the completion time of an application in the case of machines showing abnormal, we define the performance loss reduction rate (PLRR) as $(T_{regular} - T_{self-adaptive}) / T_{regular}$. $T_{self-adaptive}$ denotes the completion time of a scheduled application with the self-adaptive algorithm. $T_{regular}$ denotes the completion time of the scheduled application without the consideration of rescheduling during application execution.

Objective: Monitor the execution of an application in a set of machines and reallocate subtasks of the application if necessary.

Begin
Repeat
Measure the prediction error of the system utilization, PU_k, on machine M_k.
If $PU_k > Threshold$ **then**
  Calculate $E(T_{original})$;
  List a set of machines that are current lightly loaded, $\{M_1, M_2, ..., M_q\}$;
  If reallocation is one-to-one **then**
    For each machine, M_i, suppose it is the machine which subtasks on M_k will be assigned.
      Calculate $E(T_{reassign})^i$ with formula (2)
    End For
    Find the machine M_j, which has the maximum $E(T_{original}) - E(T_{reassign})^j$
    If $E(T_{original}) - (E(T_{reassign})^j > 0$ **then**
      Migrate the subtask on machine M_k to machine M_j.
    End If
  End If
  If reallocation is one-to-all **then**
    Sort the list of idle machines in a decreasing order with $(1-\rho_k)\tau_k$,
    Use bi-section search to find appropriate machine set *P*, which has the maximum $E(T_{original}) - E(T_{reassign})^P$
    If $E(T_{original}) - (E(T_{reassign})^P > 0$ **then**
      Migrate subtasks on machine M_k to the machine set P.
    End If
  End If
End If
Until the application is completed
End

Fig. 4. Self-adaptive task scheduling algorithm.

IV. Experiment Results

To verify the efficiency of the proposed scheduling algorithms, we compared GHS scheduling with the AppLeS scheduling, a currently widely used task scheduling system in Grid computing. Experiments were conducted to test the performance of AppLeS task scheduling and GHS task scheduling mechanism for long-term applications. The results show the proposed scheduling system outperforms AppLeS for scheduling long-running applications and it requires less number of resources running the scheduled application than AppLeS. We then examined the efficiency of the self-adaptive scheduling algorithm. Simulation results show the self-adaptive algorithm can identify the abnormality of resources

and reduce the make-span of the scheduled application through dynamically reassigning subtasks during the application execution. Finally we investigated different factors that affect the selection of an appropriate utilization threshold.

A. Comparison of GHS scheduling and AppLeS scheduling

AppLeS is probably the best known scheduling system in the Grid computation environment. It is based on NWS prediction system, which monitors and forecasts the resource performance on-line. AppLeS makes scheduling decision by estimating the mean of short-term subtask execution time based on resource availability provided by NWS. The effects of other system specific factors on the application completion time are not analyzed due to the inherent limitation of its methods. Our task scheduling system provides long-term application-level performance prediction in a heterogeneous non-dedicated distributed computation environment. The effects of machine utilization, computing power, local job service, and parallel processing on the completion time of parallel task are individually identified.

To compare the performance of AppLeS task scheduling and the proposed task scheduling for long-term applications, we conducted experiments to test the completion time of applications under each task scheduling system. Here we consider the parameter sweep application because AppLeS doesn't support optimal parallel processing. Our initial experimental results have demonstrated that the System-level Predictor in our system outperforms NWS for long-term system-level prediction although NWS is still an effective tool for what it was designed, short-term prediction. To remove the performance loss caused by NWS prediction, we let AppLeS make scheduling decision based on the prediction provided by our system. In this way, we focus on the comparison of the scheduling algorithm performance of both systems. The comparison of the application completion time (seconds) and the number of machine set with the two different scheduling systems is given in Table 1. The experiment results show that the application completion time with our proposed scheduling system is reduced by 10%-20% compared with that with AppLeS system while only about one half of machines used in AppLeS system are required for task scheduling in our system. It indicates that the GHS scheduling system can identify a smaller set of machines to solve a large application in a shorter time.

TABLE I
COMPARISON OF TASK COMPLETION TIME AND THE NUMBER OF MACHINES USED BY APPLES TASK SCHEDULING SYSTEM AND OUR TASK SCHEDULING SYSTEM

Workload (Maximum machine number)		13801.7 (25)	27619.2 (50)	53779.5 (100)	108642.5 (200)	215141.0 (400)
GHS	Task completion time (s)	496.4	557.7	712.8	874.5	1140.4
	Number of machine set	13	26	57	99	113
AppLeS	Task completion time (s)	547.4	637.4	818.3	1022.7	1266
	Number of machine set	25	50	100	200	400

B. Self-adaptive scheduling

We discuss the self-adaptive scheduling algorithm and three performance metrics in section 3.4.2. A simulation environment was set up to evaluate the rescheduling algorithm in a non-dedicated heterogeneous computation system. In our simulation environment, the arrival rate of local jobs on each machine follows Poisson distribution. The local jobs' lifetime is simulated with $2.0/x$ [24], which follows the observation of real-life processes in [25]. x is a random number between 0 and 1. The local job arrival rate and the job service rate on each machine are randomly generated in an adjustable range. The resource utilization of each machine is thus different. Each machine is simulated with different computing capacity, randomly generated between 1 and 5. In our experiment, the utilization of a normal machine is located in a certain range and the utilization of an abnormal machine is in another certain range. Both ranges are adjustable. During the execution of the application, we randomly select some machines and "mute" their local job arrival rates and service rates to make them become abnormal machines. The number of machines showing abnormality during the application execution could be also adjusted.

1) Self-adaptive scheduling of meta-tasks: A series of simulations have been conducted to verify the capability of the self-adaptive scheduling algorithm identifying normal and abnormal machines and the efficiency of the self-adaptive algorithm in term of performance loss reduction rate. The performance metrics were measured during these simulations. We studied RIAM, RINM and PLRR of the self-adaptive task scheduling with different monitor periods, different utilization thresholds, and different resource availabilities.

Table 2 give measured RIAM, RINM and PLRR with different utilization thresholds for two system monitor periods, one hour and two hours. The meta-task to be scheduled is composed of 4000 subtasks. The maximum workload of a subtask is 4000 seconds (the computation time on the slowest machine). The average local job service rate is 10 for both normal and abnormal machines. The arrival rate of local jobs on normal machines is between 0.015 and 0.04. Thus the utilization of a normal machine is between 15% and 40%. The arrival rate of local jobs on abnormal machines is between 0.065 and 0.085. So the utilization of an abnormal machine is between 60% and 85%. The size of the machine set is 20. Two machines are randomly selected to become irregular from a random time during the application running. For each parameter setting, we run simulation 30 times.

The simulation results show that RIAM decrease with the increase of utilization threshold whereas the RINM increase with the increase of utilization threshold. When the utilization threshold is low (5% or 10%), the self-adaptive scheduling algorithm can identify all abnormal machines. Meanwhile, many normal machines are identified as abnormal machines. When the utilization threshold is large (50% or 60%), none of normal machines are identified as abnormal machines. However, some of abnormal machines are not identified. This

TABLE II
THE MEASURED RIAM, RINM AND PLRR OF META-TASK RESCHEDULING WITH DIFFERENT UTILIZATION THRESHOLDS WHEN THE MONITOR PERIODS IS TWO-HOUR

Utilization threshold	RIAM (aver)	RINM (aver)	PLRR (aver)	PLRR (min)	PLRR (max)
5%	1.0	0.01	0.24	-0.28	0.49
10%	1.0	0.56	0.24	-0.33	0.61
20%	1.0	1.0	0.51	0.0	0.74
30%	0.93	1.0	0.47	0.0	0.71
40%	0.78	1.0	0.41	0.0	0.71
50%	0.51	1.0	0.29	0.0	0.62
60%	0.31	1.0	0.17	0.0	0.17

TABLE III
THE MEASURED RIAM, RINM AND PLRR OF META-TASK RESCHEDULING WITH DIFFERENT UTILIZATION THRESHOLDS WHEN THE MONITOR PERIOD IS ONE-HOUR

Utilization threshold	RIAM (aver)	RINM (aver)	PLRR (aver)	PLRR (min)	PLRR (max)
5%	1.0	0.00	-0.38	-0.53	0.59
10%	1.0	0.07	0.25	-0.16	0.61
20%	1.0	0.98	0.48	0.0	0.72
30%	0.98	1.0	0.53	0.0	0.75
40%	0.91	1.0	0.46	0.0	0.75
50%	0.73	1.0	0.37	0.0	0.65
60%	0.48	1.0	0.25	0.0	0.62

TABLE IV
THE MEASURED RIAM, RINM AND PLRR OF META-TASK RESCHEDULING WITH DIFFERENT UTILIZATION THRESHOLDS WHEN THE MONITOR PERIOD IS FOUR-HOUR

Utilization threshold	RIAM (aver)	RINM (aver)	PLRR (aver)	PLRR (min)	PLRR (max)
5%	1.0	0.07	-0.28	-1.25	0.48
10%	1.0	0.92	0.47	0.0	0.73
20%	0.98	1.0	0.48	0.0	0.71
30%	0.91	1.0	0.45	0.0	0.71
40%	0.70	1.0	0.37	0.0	0.65
50%	0.43	1.0	0.24	0.0	0.62
60%	0.17	1.0	0.1	0.0	0.62

TABLE V
THE MEASURED RIAM, RINM AND PLRR OF META-TASK RESCHEDULING WITH DIFFERENT UTILIZATION THRESHOLDS WHEN THE MONITOR PERIOD IS HALF-AN-HOUR

Utilization threshold	RIAM (aver)	RINM (aver)	PLRR (aver)	PLRR (min)	PLRR (max)
5%	1.0	0.0	0.28	0.0	0.80
10%	1.0	0.0	0.12	-0.14	0.66
20%	1.0	0.98	0.02	-0.75	0.51
30%	1.0	0.97	0.49	0.0	0.73
40%	0.98	0.99	0.49	0.0	0.75
50%	0.91	1.0	0.48	0.0	0.71
60%	0.73	1.0	0.39	0.0	0.66

leads the reduction of PLRR.

From Table 2 and Table 3, we can find that the self-adaptive algorithm efficiently reduce the application performance loss in most situations although the application performance may get worse in a few extreme situations. It works best when the utilization threshold is set between 20%-30%. When the utilization threshold is out of this range, either low or high, the gain is comparably smaller. We notice that in this appropriate utilization threshold area, both of RIAM and RINM are close to 1.0, which indicate that the self-adaptive scheduling algorithm works best when it can rightly identify both the normal machines and the abnormal machines.

We also measured RIAM, RINM and PLRR when the system monitor periods are set as four hours and half an hour. We keep other simulation parameter setting the same as the above. The simulation results are illustrated in Table 4 and Table 5. We find that in both situations the self-adaptive algorithm benefits the application performance. In the former situation, the PLRR is better when the utilization threshold is 10% or 20% while the PLRR is better when the utilization threshold is 30% or 40% for the latter. This indicates that when we increase the system monitor period, even a small threshold could benefit the application performance. If the system monitor period is short, for example, half an hour, we have to choose a high utilization threshold so that the self-adaptive task scheduling algorithm can efficiently reduce the performance loss caused by machine mutation.

Is there any other factor that affects the selection of an appropriate utilization threshold besides the system monitor period? We conducted a series of simulations to study this problem. We tested the efficiency of the self-adaptive scheduling algorithm with different resource availability variation and various utilization differences between abnormal machines and normal machines. We observed different appropriate threshold range in these simulations. However we find that both RIAM and RINM are close to 1.0 for all appropriate threshold ranges.

Fig. 5 gives the PLRR of meta-task scheduling under various average utilization differences between abnormal machines and normal machines (20%, 30%, 40%, 50%). For each utilization difference, we measured the PLRR in four utilization thresholds, 0.1, 0.2, 0.3, and 0.4. The monitor period is two hours. The PLRR is the average of 30 times of running simulation. The simulation results show that the

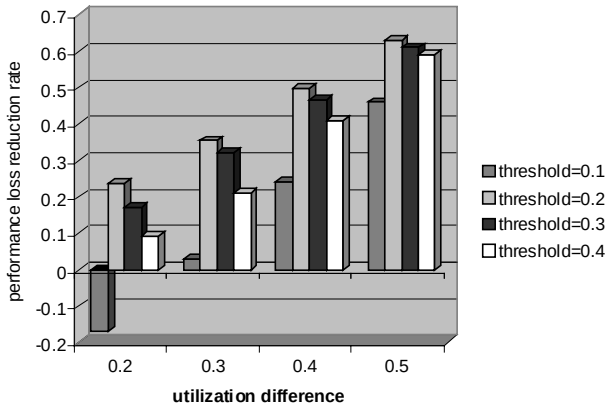

Fig. 5. The measured PLRR of meta-task rescheduling with various average resource utilization differences.

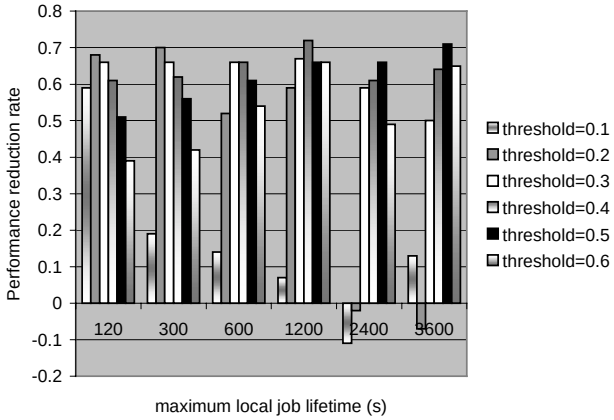

Fig. 6. The measured PLRR of meta-task rescheduling with various local job lifetime variances.

difference of PLRRs under four thresholds is getting smaller when the average utilization difference increases. We can find when the average utilization difference is 50% the PLRR is good even the threshold is set as 0.1. This indicates that the more the average utilization difference between abnormal machines and normal machines, the wider the appropriate threshold range. The simulation results also show that there is a constant improvement of PLRR for each threshold with the increase of utilization difference. We find that the PLRR increase around 40% when we increase the average utilization difference from 20% to 50% when the utilization threshold is set as 20%.

In our experiment, we simulated local jobs' lifetime with 2.0/x where x is a randomly generated number between 0 and 1. [24]. This distribution follows the observation of real-life processes [25]. Let Z denotes the maximum lifetime of local jobs. The variance of job lifetime is calculated as $\frac{2Z}{Z-2}(Z-1)-((\frac{2Z}{Z-2})(\ln Z-\ln 2))^2$. The more Z is, the bigger variance of job lifetime and thus the variance of resource utilization. Fig. 6 gives the measured PLRR with various local job lifetime variances. We set the maximum lifetime of local jobs from 120s to 3600s. The simulation results show that with the increase of job lifetime variance, the appropriate threshold range tends to move from the low level to the high level. When the maximum lifetime of local jobs is small (120s), the appropriate threshold is between 0.1 and 0.4. When the maximum lifetime of local jobs is large (3600s), the appropriate threshold is between 0.4 and 0.6. This simulation indicates that in the situation of high variance of resource availability, a large utilization threshold is preferred for a better performance gain.

2) Self-adaptive scheduling of a parallel program: We evaluate the efficiency of the self-adaptive scheduling algorithm for parallel programs. Table 4 gives the measured RIAM, RINM and PLRR of parallel program rescheduling with different utilization thresholds when the monitor periods are one hour and two-hours respectively. We used the same simulation setting as above. The utilization of a normal machine is between 15% and 40%. The utilization of an abnormal machine is between 60% and 85%. We observed the same results as previous experiments for meta-task scheduling. The simulation results show that the self-adaptive algorithm can effectively reduce the application performance loss and it works best when RIAM and RINM are close to 1.0. In Table 6 and Table 7, we find that the self-adaptive scheduling algorithm introduces more performance loss for parallel program rescheduling than meta-task rescheduling when the utilization threshold is set as a small value, such as 0.1. This reason is that because in the process of rescheduling a parallel program we do not consider further partition the workload of the subtask on the identified abnormal machine (we apply one-to-one reallocation), the performance loss caused by wrong identification of abnormal machine is bigger than that of a meta-task where the subtasks on the wrong identified abnormal machine will be distributed to all possible normal machines. We also investigated the relationship among the appropriate threshold range, the average utilization difference between normal machines and abnormal machines, and the resource utilization variance for parallel program rescheduling. We observed the similar results as meta-task rescheduling.

TABLE VI
THE MEASURED RIAM, RINM AND PLRR OF PARALLEL PROGRAM RESCHEDULING WITH DIFFERENT UTILIZATION THRESHOLDS WHEN THE MONITOR PERIOD IS TWO-HOUR

Utilization threshold	RIAM (aver)	RINM (aver)	PLRR (aver)	PLRR (min)	PLRR (max)
5%	1.0	0.00	-0.23	-6.35	0.49
10%	1.0	0.38	0.11	-3.92	0.65
20%	1.0	0.99	0.40	0.0	0.67
30%	1.0	1.0	0.36	-0.67	0.67
40%	0.93	1.0	0.35	-0.67	0.64
50%	0.76	1.0	0.18	-1.79	0.64
60%	0.43	1.0	0.08	-0.66	0.64

TABLE VII
THE MEASURED RIAM, RINM AND PLRR OF PARALLEL PROGRAM RESCHEDULING WITH DIFFERENT UTILIZATION THRESHOLDS WHEN THE MONITOR PERIOD IS ONE-HOUR

Utilization threshold	RIAM (aver)	RINM (aver)	PLRR (aver)	PLRR (min)	PLRR (max)
5%	1.0	0.00	-0.61	-7.49	0.42
10%	1.0	0.02	-0.64	-8.19	0.48
20%	1.0	0.91	0.38	0.0	0.67
30%	1.0	0.99	0.36	-0.70	0.68
40%	0.93	1.0	0.36	-0.61	0.68
50%	0.76	1.0	0.10	-3.42	0.64
60%	0.43	1.0	0.05	-0.84	0.65

V. CONCLUSION AND FUTURE WORK

In this paper, we study task scheduling of parallel programs and meta-tasks in a heterogeneous shared environment. We present the system design of a general-purpose task scheduling system. A task group allocation method and an optimal scheduling algorithm are introduced for meta-task scheduling. To reduce performance loss caused by possible machine "mutation", a self-adaptive task scheduling algorithm is developed and three performance metrics are proposed. Initial experimental tests were conducted on the Sun ComputeFarm at IIT to compare GHS task scheduling system with other current performance prediction based task scheduling system. The results show that the proposed scheduling system provides an appropriate general-purpose scheduling mechanism for long-term applications. The dynamic and self-adaptive scheduling algorithm we proposed adequately captures the dynamic nature of distributed computing and therefore provides a robust scheduling by reallocating tasks in the presence of resource abnormalities, which is not addressed by current scheduling systems. The performance loss can be reduced by 50%-60%. We have investigated several factors that affect the selection of an appropriate threshold. The experiment results demonstrate that GHS scheduling system outperforms current systems in scheduling long-term applications in a homogeneous or heterogeneous computing environment. The task completion time with our scheduling system decreases by 10%-20% compared with that with AppLeS scheduling system while only about one half of machines used in AppLeS system are required for application running in the proposed scheduling system.

We proposed and implemented a prototype of long-term, application-level task scheduling system for shared heterogeneous computing environment based on probabilistic approaches. It is a satisfactory complement of existing performance evaluation and task scheduling tools. The three components of our task scheduling, task allocator, scheduler and predictor, can be easily integrated into existing toolkits for better service. To further improve the applicability and accuracy of the proposed task scheduling system, the communication and the migration cost in task allocation will be investigated. We also want to study the task scheduling of multiple applications with the consideration of QoS in the future.

ACKNOWLEDGMENT

This research was supported in part by National Science Foundation under NSF grant EIA-0130673, ANI-0123930, and by Army Research Office under ARO grant DAAD19-01-1-0432.

REFERENCES

[1] Michael Litzkow, Miron Livny, and Matt Mutka, "Condor - a hunter of idle workstations," *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104-111, June 1988.

[2] Henri Casanova and Jack Dongarra, "NetSolve: a network server for solving computational science problems," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 3, pp. 212-223, Fall 1997.

[3] Abramson D., Sosic R., Giddy J., and Hall B., "Nimrod: a tool for performing parametised simulations using distributed workstations," *The 4th IEEE Symposium on High Performance Distributed Computing*, Virginia, Aug. 1995.

[4] Ian Foster and Carl Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. ISBN 1-55860-475-8, July 1998.

[5] H. Dail, H. Casanova, and F. Berman, "A decoupled scheduling approach for the GrADS environment," *Proceedings of SC 2002*, Baltimore, Nov. 2002.

[6] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonindentical processors," *Journal of the ACM*, 24(2): 280-289, Apr. 1977.

[7] Ian Foster and Adriana Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," *2nd International Workshop on Peer-to-Peer Systems*, Feb. 2003.

[8] Fran Berman, Richard Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao, "Application-level scheduling on distributed heterogeneous networks," *Supercomputing'96*, Nov. 1996.

[9] X. -H. Sun and M. Wu, "A performance prediction and task scheduling system for grid computing," *Proc. of 2003 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, Apr. 2003.

[10] Henri Casanova, Graziano Obertelli, Francine Berman, and Rich Wolski, "The AppLeS parameter sweep template: user-level middleware for the grid," *Proceedings of SC 2000*, Nov. 2000.

[11] Y. Yang and H. Casanova, "UMR: a multi-round algorithm for scheduling divisible workloads," *Proceedings of the International Parallel and Distributed Processing Symposium*, Nice, France, Apr. 2003.

[12] I. Foster, A. Roy, and V. Sander, "A quality of service architecture that combines resource reservation and application adaptation," *International Workshop on Quality of Service*, pp. 181-188, June 2000.

[13] Y. A. Li and J. K. Antonio, "Estimating the execution time distribution for a task graph in a heterogeneous computing system," *6th Heterogeneous Computing Workshop*, Geneva, Switzerland, Apr. 1997.

[14] Steve Chapin, Dimitrios Katramatos, John Karpovich, and Andrew Grimshaw, "The legion resource management system," *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99)*, San Juan, Puerto Rico, April 1999.

[15] Henri Casanova, MyungHo Kim, James S. Plank, and Jack Dongarra, "Adaptive scheduling for task farming with grid middleware," *The International Journal of High Performance Computing*, vol. 13, no. 3, pp. 231-240, Fall 1999.

[16] Richard Wolski, "Dynamically forecasting network performance using the network weather service," *Journal of Cluster Computing*, vol. 1, no. 1, pp. 119-132, Jan. 1998.

[17] Richard Wolski, Neil T. Spring, and Jim Hayes, "The network weather service: a distributed resource performance forecasting service for

metacomputing," *Journal of Future Generation Computing Systems*, vol. 15, no. 5-6, pp. 757-768, Oct. 1999.

[18] Linguo Gong, Xian-He Sun, and Edward F. Waston, "Performance modeling and prediction of non-dedicated network computing," *IEEE Trans. on Computer*, vol. 51, no 9, Sept. 2002.

[19] F. Berman, R. Wolski, H. Casanova, W. Cirne, et al. "Adaptive Computing on the Grid Using AppLeS," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 369--382, 2003.

[20] P. Dinda and D. O'Hallaron, "An extensible toolkit for resource prediction in distributed systems," Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July 1999.

[21] Mutka, M. and M. Livny, "The available capacity of a privately owned machine environment," *Performance Evaluation*, vol. 12, no. 4, pp. 269-284, 1991.

[22] Remzi H. Arpaci, Andrea C. Dusseau, Amin M. Vahdat, Liu, Lok T., Anderson, Thomas E., and Patterson, David A., "The interaction of parallel and sequential workloads on a network of machines," *Proc. of ACM SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, pp. 267-278, May 1995.

[23] A. Acharya, G. Edjlali, and J. Saltz, "The utility of exploiting idle workstations for parallel computation," *Proc. SIGMETRICS*, pp. 225-236, June 1997.

[24] Yair Amir, Baruch Awerbuch, Amnon Barak, R. Sean Borgstrom, and Arie Keren, "An opportunity cost approach for job assignment in a scalable computing cluster," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, July 2000.

[25] M. Harchol-Balter and A. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *Proc. ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems*, 1996.

Ming Wu is currently a Ph.D. candidate in the Department of Computer Science at Illinois Institute of Technology. He received his bachelor of Engineering from Xidian University, China, in 1994 and the Master degree of Science from the University of Science and Technology of China, in 1997. His research focuses on the design and development of performance evaluation and task scheduling systems for distributed computing environment, with specific interests in Grid computing.

Xian-He Sun received the BS degree in mathematics from Beijing Normal University, Beijing, China, in 1982, the MS degree in mathematics, and the MS and PhD degrees in computer science from Michigan State University, East Lansing, in 1985, 1987, and 1990, respectively. He was a staff scientist at ICASE, NASA Langley Research Center and was an associate professor in the Computer Science Department at Louisiana State University, Baton Rouge. He has been serving as a faculty member of the Computer Science Department at the Illinois Institute of Technology (IIT), Chicago, since 1999. Currently, he is a professor and the director of the Scalable Computing Software Laboratory in the Computer Science Department at IIT, and is a guest faculty member at the Argonne National Laboratory. Dr. Sun's research interests include grid and cluster computing, software system, pervasive computing, performance evaluation, and scientific computing. He has published intensively in the field and his research has been supported by DoD, DoE, NASA, US National Science Foundation, and other government agencies. He is a senior member of the IEEE, a member of the ACM, New York Academy of Science, PHI KAPPA PHI, and has served and is serving as the chairman or on the program committee for a number of international conferences and workshops, including current service as the general co-chair of the Grid and Cooperative Computing (GCC03) workshop, area chair of the technical committee of the IEEE SuperComputing (SC03) conference, vice chair of the programming committee of the International Conference on Parallel Processing (ICPP04), and vice president of the Society of Chinese American Professors and Scientists. He received the ONR and ASEE Certificate of Recognition award in 1999, and received the Best Paper Award from the International Conference on Parallel Processing (ICPP01) in 2001.