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

Ming Wu, Xian-He Sun
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60616, USA
{wuming, sun}@iit.edu

Abstract

The efforts to construct a national scale grid computing environment has brought unprecedented computing capacity. Exploiting this complex infrastructure requires efficient middleware to support the execution of a distributed application, composed of a set of subtasks, for best performance. This presents the challenge how to schedule these subtasks in shared heterogeneous systems. Current work has several limitations. Most scheduling systems are based on determined estimation of task completion time. Current application-level scheduling algorithms are too closely coupled with application internal structures. The application performance may suffer when some resources represent abnormal usage pattern during applications execution. To address these issues, we develop a prototype of Grid Harvest Service (GHS) to provide dynamic and self-adaptive task scheduling. Experimental results show GHS outperforms current systems in scheduling large applications in a non-dedicated heterogeneous environment.

1. Introduction

In order to provide high performance computation power to serve the increasing need of large applications, people strive to improve the single machine's capacity or construct a distributed system composed of a scalable set of machines. Compared to the former, in which improvement is mainly up to hardware technology development, the construction of a distributed system for resource collaboration is more complex. Some of well-known existing or ongoing distributed systems composed of heterogeneous resources are Condor [1], NetSolve [2], Nimrod [3], Globus and the Grid computation environment [4]. While these systems, especially the Grid, bring unprecedented computing power for users, how to deliver this great computing power is still an elusive problem. One of the major puzzles is how to schedule applications in these systems. In general, scheduling applications in a distributed system is a NP-hard problem. Many heuristic scheduling algorithms and systems are proposed to address this issue. However, most of scheduling algorithms so far proposed are for dedicated systems. By dedicated, we refer resources in the system serve the application in a dedicated fashion. This is against the situation that most of current distributed systems are resource-sharing environments [5].

Good schedules in a shared environment involve the integration of application specific information and system specific information. We study 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 and job arrival rate, and application-specific information, such as workload, divisibility, parallel processing, on the application performance have been identified. Our preliminary results demonstrate the effectiveness of our scheduling mechanism for long-term applications. Our earlier work [6] 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 applications, the parameter-sweep applications. It is composed of a set of independent and indivisible tasks [7].

A key question is that how to maintain the parallel application performance when some of resources assigned for subtasks represent abnormal status from their historical information. A successful scheduling depends on the accuracy of system status prediction, which is deducted from the resource historical information. The application performance suffers under the situation of resource abnormality and there is a fair chance that a resource may show different usage pattern from its history in a large distributed system. To provide a robust task scheduler to work in this dynamic situation, we introduce a self-adaptive task scheduling algorithm, which monitors the long-running application progress and detects possible

resource abnormality. The self-adaptive scheduling algorithm will select appropriate resources and reassign tasks on abnormal machines to these selected resources based on application-level prediction.

Our goal in task scheduling system is to provide a general-purpose and robust scheduling approach. The task scheduling is divided into three parts, allocator, predictor and scheduler. Allocator decides how to allocate subtasks of a divisible application for each machine in a group of machines, where a divisible application refers to an application that could be partitioned into a set of subtasks. Predictor estimates the application execution time distribution on each machine. Scheduler decides which set of machines is the best among all sets of machines.

In the following sections, we will show that different task allocation methods and scheduling algorithms are used for different situations. Even for the same situation, different task allocation approaches could be applied. In this way, we provide a flexible task scheduling mechanism for the need of different application scheduling scenarios. Our proposed scheduling system is much flexible as, on one hand it could be easily integrated with other scheduling systems, while on the other hand other 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. We then proceed to study the system-level prediction and application-level prediction. Various scenarios of task allocation and scheduling are then discussed. Experimental results are presented in Section 4. We compare GHS scheduling system with current scheduling systems and verify the efficiency of the self-adaptive scheduling algorithm. Finally we conclude and summarize our work in section 5.

2. Related work

Related earlier work in task scheduling policy mainly focused on dedicated systems, which cannot be assumed in non-dedicated computing environment like Grid. A resource reservation strategy [8] is proposed to address this problem. The shared resources are reserved in advance for one user's dedicated occupancy. It is useful for time urgent application and could be applied on space-shared supercomputers because they can provide dedicated nodes for running the applications. However, it is not feasible on time-shared resources such as PC and workstation in a heterogeneous system because private owners usually don't want their machine dedicated for grid tasks. Task scheduling in such a shared non-dedicated computing environment offers a big challenge. A task scheduling mechanism based on determined prediction of machine availability has been investigated in [7]. This 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 distributed applications in a large system since the prediction error caused by 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 individual task is pre-known [9]. 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) [8] is a subsystem of Globus project. It provides mechanism for resource reservation so that applications can receive a certain level of service from a resource. As we mentioned before, this policy doesn't favor the privilege of the private owners of shared resources. Condor system [1] provides a matchmaking mechanism to allocate resources with ClassAds. Legion system [10] also supports resource reservation. It focuses on providing basic mechanisms for building application-level scheduling algorithms rather than construct scheduling algorithm itself. A simple random selection policy is provided as the default scheduling mechanism. Currently, the scheduling algorithms in the AppLeS [8] project are supported by the short-term system prediction provided by NWS services. Although they consider the availability of resources in making the scheduling decision, none of them fully identify the effect of "sharing" characteristics of resources on the execution time of applications. Their scheduling algorithms are too closely coupled with application internal structures They don't consider rescheduling during the application execution when some resources show abnormality.

3. Task scheduling system

Prediction of application and system performance is necessary for a good schedule. Some of current scheduling systems did 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 GHS is based on probabilistic modeling. The effect of system-specific information on application performance has been identified by our general performance model [11]. How to apply the prediction for task scheduling so that the make span of applications could be minimized is our goal. The application-level prediction and related measurement mechanism has been discussed in [6].

In this study, a system-level prediction approach has been introduced to extend our work in prediction. A task allocation component is separated from scheduling and different allocation methods are discussed. By dividing task scheduling into three parts, task allocator, predictor and task scheduler, a general task scheduling mechanism has been provided. It could be easily integrated by other scheduling system and we can also easily introduce other people's work into GHS. To reduce the performance loss brought by abnormal resource usage pattern, an adaptive scheduling algorithm is investigated. In this paper, we focus on scheduling computation-intensive applications. Other people's work in communication cost estimation [12] could be introduced into GHS. More efforts will be made in considering communication-intensive applications in the future.

3.1 System architecture

A block diagram of GHS system design is shown below in **Figure 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 which could be provided by 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 tasks among resources based on the prediction of system status provided by system-level predictor. The map of tasks on machines will be forwarded to Application-level Predictor to estimate the application performance. A 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 (Network sensors, IO sensors and CPU sensors) and stored in the Resource Information Service, which provides the access of resource information for the System-level Predictor and the Task Scheduler.

3.2 Predictor

Based on the observation of machine usage pattern [13,14], we assume that the local job processing follows M/G/1 queuing system, which is determined by system parameters λ, ρ, σ, where ρ is the system utilization, λ is the arrival rate of the local jobs and σ is the standard deviation of service time of local jobs. These parameters may vary throughout a day in a real system. How to capture these dynamic characteristics of resources is the primary goal of the system-level predictor.

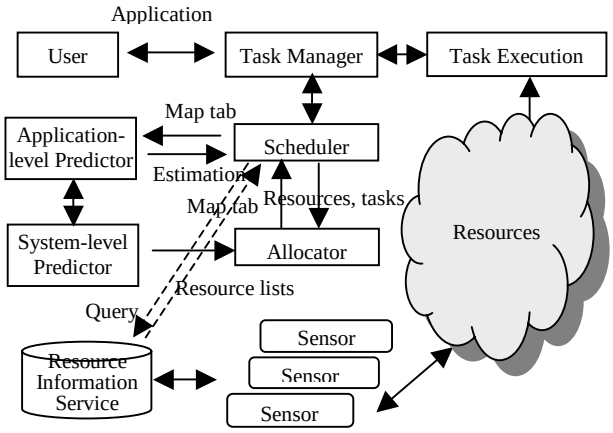

Figure 1. A framework of GHS task scheduling system

To estimate the values of system parameters in the incoming time interval, we basically choose the mean-based method, which uses arithmetic average of measurement of a parameter as an estimate of its mean value. Similar methods are used in NWS and RPS project. The difference lies in the selection of the sample space. Instead of using only the average of measurements of a system parameter during the previous N number of continual time intervals as the estimate of the system parameter value for the next time interval, we also consider the average of measurements during the previous N number of "similar" time intervals. For example, if we predict a system parameter for the time period from 5 pm to 7 pm, the average of parameter values measured during the continuous previous 2 hours periods before 5 pm and the average of parameter values measured from 5 pm to 7 pm in previous days are both collected for estimation. We notice that other forecast models, such as LAST model, AR model, and ARIMA model, can also be used in our algorithm.

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 other people's work is that we estimate the application make-span based on probabilities analysis instead of determined approach used by other scheduling systems. So the effect of both system specific information and application-specific information on the application performance is identified in our application-level prediction. These system parameters, λ, ρ and σ, reflect the sharing characteristics of resources in a non-dedicated distributed system. The workload and divisibility of an application reflect the general characteristics of applications. If an application is a single indivisible task, the cumulative distribution function of the

application completion time on a machine can be calculated as [SuWu03]:

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

where $U(S)$ is the sum of the busy periods of the machine owner's local jobs. We denote τ the computing capacity of the machine and w the workload of the application. If the workload of the 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). We assume that the sub-workload on each machine, w_k , is pre-known before running application-level predictor. The calculation of w_k is actually finished by the Task Allocator component.

3.3 Task allocator

When an application is a single indivisible task, we choose a machine where the sum of the expectation and variance of application execution time is minimal. In general, however, an application in a distributed system is likely to be solved concurrently for best performance. This may happen in two scenarios. The first is that the application is divisible, which means the application can be partitioned into subtasks arbitrarily. Another scenario is that the application consists of independent indivisible subtasks. We cannot further partition these indivisible subtasks. There are still two situations generally to be concerned. One is that subtasks have no dependency among each other. We define such an application as a meta-task. Another situation is that there are dependencies among subtasks. In this paper, we study the meta-task scheduling. An ideal example of such applications is the class of parameter-sweep applications, which is composed of a set of independent tasks [7].

For a given set of resources, the Task Allocator decides how to partition/group subtasks of an application and then map them to a set of resources to reach optimal performance. If the workload of an application is divisible, the task allocation is a partition problem. If an application is composed of indivisible subtasks, 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 based on the expected mean execution time of subtasks on the resource. **Figure 2** gives the details of mean-time task allocation algorithm where ρ_k is the utilization of resource *k* and τ_k its processing power.

When an application is composed of indivisible subtasks, the Task Allocation component uses the min-min algorithm to group subtasks and map each subtask group to one of resources for optimal performance based on the estimation of subtask execution time. Suppose we have an application composed of a number of independent tasks, $\{T_1, T_2, ..., T_p\}$ with workload $\{w_1, w_2, ..., w_p\}$ and a list of machines $\{M_1, M_2, ..., M_q\}$, **Figure.3** shows the min-min task group allocation algorithm in details.

Assumption: a parallel task with workload *W*, and a list of resources $\{M_1, M_2, ..., M_q\}$.

Objective: map subtask w_k to each machine $\{M_1, M_2, ..., M_q\}$.

Begin

For each machine M_k $(1 \le k \le q)$

$$w_k = \frac{W}{\sum_{k=1}^{m} (1 - \rho_k)\tau_k} (1 - \rho_k)\tau_k$$

End For

Return w_k $(1 \le k \le q)$;

End

Figure 2. Mean-time task partition algorithm

3.4 Self-adaptive 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. Our Task Scheduler supports different scheduling scenarios according to the application characteristics. Our Task Scheduler can support different task scheduling scenarios: single sequential task scheduling, optimal parallel processing with a given number of subtasks, optimal parallel processing and meta-task scheduling. A heuristic task-scheduling algorithm is proposed to find an acceptable solution with a reasonable cost. In this study, we focus on 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 of resources running subtasks of the scheduled application is degraded, the completion time of application will increase. How to decide whether we should reschedule the application and which resource we should choose are the problems we need to consider. 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. The subtasks could be either migrated or restarted on the selected machine, which is up to migration cost and whether migration is possible. The

Assumption: an application 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

Figure 3. Min-min task group allocation algorithm

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 performance gain for the application performance. Because there is no general performance model to estimate migration cost and the communication cost of process migration may be overlapped with the computation of subtasks, we ignore the effect of migration cost in the rescheduling decision. The consideration of migration cost is our future work.

The estimate of application performance is based on the measurement of system parameters. The details of measure mechanism have been given in our previous work [SuWu03]. We can use the same technique 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

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 $PU_k > Throttle$ **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 the subtask will be assigned. Calculate $E(T_{reassign})^i$ with formula (2)

End For

Find the machine M_i, 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_i.

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 the subtask on machine k to the machine set P.

End If

End If

Until the application is completed

End

Figure 4. Self-adaptive task scheduling algorithm

system time will be used as the estimate of ρ_k.

When the system parameters indicate the abnormality of resources, we need to estimate the application performance in the new situation. An extension of formula (1) is used to calculate the cumulative distribution function of the application execution time in this situation.

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

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 throttle, we assume the resource suffers abnormality. The set of the monitor period is related to the set of the throttle. In the next section, we conducted experiments with different monitor periods and throttles. Using formula (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 $E(T_{original})$ and the later $E(T_{reassign})$. 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 performance gain if we assign subtasks on an abnormal resource to them. A detailed description of the self-adaptive scheduling algorithm is given in **Figure 4**. In this algorithm, we assume 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 tasks 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 3.2 to map the uncompleted tasks on an abnormal machine into other machines. We set $W_k = 0$, $TG_k = \phi$ for the identified abnormal machines and $W_k =$ {the sum of workload of completed tasks} and $TG_k =$ {the set of completed tasks} for other machines.

4. Experiment results

To verify the efficiency of our scheduling algorithms, we compare GHS schedule with the AppLeS schedule, a currently widely used task scheduling mechanism in Grid computing. AppLeS schedule is based on NWS short-term system prediction. Experiments are conducted to test the performance of AppLeS task scheduling and GHS task scheduling for long-term applications. The results show GHS outperforms AppLeS for scheduling long-running applications and it requires less number of resources running the scheduled application than AppLeS. We then test the efficiency of our 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.

4.1 Comparison of GHS schedule and AppLeS schedule

AppLeS makes scheduling decision by estimating the mean of short-term subtask execution time based on the availability of resources. The effects of other system specific factors are not analyzed due to the inherent limitation of these methods. GHS task scheduling system provides long-term application-level performance prediction in 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 test the performance of AppLeS task scheduling and GHS task scheduling for long-term applications, we conduct experiments to compare the completion time of applications based on AppLeS task scheduling and GHS task scheduling. Here we consider the parameter sweep application because AppLeS doesn't support optimal parallel processing. To remove the difference caused by system-level prediction, we assume that the system parameters are pre-known by both scheduling systems. The comparison of 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 GHS 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 GHS. That indicates that GHS scheduling system has the real potential to identify a smaller set of machines to execute long-running applications in a shorter time.

4.2 Self-adaptive scheduling

In section 3.4, we have discussed how to reschedule when we find that a resource showing irregular situation. To testify the efficiency of the rescheduling algorithm, we have to answer the following questions:

1. Whether the algorithm can identify the abnormal machine? What is the error rate?
2. Whether the self-adaptive scheduling algorithm can reduce the application performance loss due to resource abnormality?
3. What is the appropriate utilization throttle to benefit the application performance?

To answer these questions, we go through a series of simulations. The arrival rate of local jobs on each machine's follows Poisson distribution. The local jobs' lifetime is simulated with 2.0/x [15]. The arrival rate and service rate are randomly generated for each machine in an

adjustable range. During the execution of the application, we randomly select some machines and change their local job arrival rates and service rates. The number of machine showing abnormality during the application execution could be also adjusted. The workload of the application is divisible.

Table 1. Comparison of task completion time and the number of machine set used by AppLeS system and GHS 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

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. So 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 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 which are actually normal machines and N_n denotes the number of normal machines. 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 $\frac{T_{regular}-T_{self-adaptive}}{T_{regular}}$. $T_{self-adaptive}$ denotes the completion time of a scheduled application with the self-adaptive algorithm. $T_{regular}$ denotes the completion time of the scheduled application with the normal compile-time scheduling. **Table 2** gives measured RIAM, RINM and PLRR with different utilization throttles. We set two system monitor periods, one hour and two hours. The utilization of a normal machine is between 15% and 40%. The utilization of an abnormal machine is between 60% and 75%. 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 monitor period setting, we run simulation 30 times. The simulation results show that RIAM decrease with the increase of utilization throttle while the RINM increase with the increase of utilization throttle. From **Table 2**, we can find that the self-adaptive algorithm efficiently reduce the application performance loss. It works best when the utilization throttle is set between 10%-30%. When the utilization throttle is out of this range, either low or high, the gain is comparably smaller.

We believe that the more the average utilization difference between abnormal machines and normal machines, the more benefit we can get through the self-adaptive scheduling algorithm. Our simulation results indicate that the PLRR increases when either the average utilization of abnormal machine increases or the average utilization of normal machines decreases. Table 3 shows measured PLRR when we set different utilization differences. The monitor period is two hours. The utilization throttle is 20%. The PLRR is the average of 30 times of running simulation. We find that the PLRR increase around 65% when we increase the utilization difference from 10% to 70%.

Table 2. The measured RIAM, RINM and PLRR with different utilization throttles

Utilization Throttle	Monitor period (two hour)			Monitor period (one hour)		
	RIAM	RINM	PLRR	RIAM	RINM	PLRR
5%	1.0	0.01	0.24	1.0	0.00	-0.38
10%	1.0	0.71	0.58	1.0	0.06	0.3
20%	1.0	1.0	0.51	1.0	0.98	0.54
30%	0.91	1.0	0.47	0.97	1.0	0.53
40%	0.63	1.0	0.31	0.85	1.0	0.44
50%	0.18	1.0	0.26	0.56	1.0	0.22

5. Conclusion and future work

In this paper, we study task scheduling of a parallel or distributed application with a divisible workload in a heterogeneous environment. We present prediction mechanisms both in system-level and application-level based on a new performance model. Different task allocation algorithms are introduced for scheduling parallel program and meta-task. To reduce performance loss caused by possible machine "mutation", a self-adaptive task scheduling algorithm is developed. Initial experimental testing was conducted at the Sun ComputeFarm at IIT to compare GHS task scheduling system with other current performance prediction based task scheduling system. The results show that GHS

provides a general-purpose scheduling mechanism for long-term applications. The dynamic and self-adaptive scheduling algorithm we propose adequately captures the dynamic nature of distributed computing and therefore provides a robust scheduling by reallocating tasks in the represent of resource abnormalities, which is not addressed by current scheduling systems. The performance loss can be reduced by 50%-60%. The experiment results demonstrate that GHS outperforms current systems in scheduling long-term applications in a heterogeneous computing environment. The task completion time with GHS 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 GHS system.

Table 3. The measured PLRR with different utilization difference

Utilization difference	10%	20%	30%	40%	50%	60%	70%
PLRR	-0.02	0.08	0.21	0.32	0.45	0.56	0.65

We propose and implement a prototype of long-term, application-level task scheduling system for shared heterogeneous computing environment based on probabilistic approach. It is a satisfactory complement of existing performance evaluation and task scheduling tools. The three parts of task scheduling, task allocator, scheduler and predictor, can be easily integrated into existing toolkits for better service. To improve the applicability and accuracy of GHS scheduling system, the communication and the migration cost in task allocation will be considered in the future work.

Acknowledgments

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*, pages 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*, Volume 11, Number 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, August 1995.

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

[5] Ian Foster, Adriana Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing", *The 2nd International Workshop on Peer-to-Peer Systems*, February 2003.

[6] 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, April, 2003.

[7] Henri Casanova, Graziano Obertelli, Francine Berman and Rich Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid", *Proceedings of Super Computer 2000*, November 2000.

[8] I. Foster, A. Roy, 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.

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

[10] Steve Chapin, Dimitrios Katramatos, John Karpovich, Andrew Grimshaw, "The Legion Resource Management System", *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, San Juan, Puerto Rico, April, 1999.

[11] 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, September, 2002.

[12] Downey, A.B., "Using pathchar to estimate Internet link characteristics", *Proc. SIGCOMM 1999*, pp. 241-250, Cambridge, MA, Sept. 1999.

[13] Remzi H. Arpaci, Andrea C. Dusseau, Amin M. Vahdat, et al, "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.

[14] Mutka, M., and M. Livny, "The Available Capacity of a Privately Owned Machine Environment", *Performance Evaluation*, Vol. 12, No 4, pp. 269-284, 1991.

[15] Yair Amir, Baruch Awerbuch, Amnon Barak, R. Sean Borgstrom, 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.