Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 66 (2006) 1322–1337
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Grid harvest service: A performance system of grid computing

Ming Wu, Xian-He Sun*

Department of Computer Science, Illinois Institute of Technology, Chicago, IL, 60616, USA

Received 6 November 2004; received in revised form 30 September 2005; accepted 9 May 2006
Available online 24 July 2006

Abstract

Conventional performance evaluation mechanisms focus on dedicated systems. Grid computing infrastructure, on the other hand, is a shared collaborative environment constructed on virtual organizations. Each organization has its own resource management policy and usage pattern. The non-dedicated characteristic of Grid computing prevents the leverage of conventional performance evaluation systems. In this study, we introduce the grid harvest service (GHS) performance evaluation and task scheduling system for solving large-scale applications in a shared environment. GHS is based on a novel performance prediction model and a set of task scheduling algorithms. GHS supports three classes of task scheduling, single task, parallel processing and meta-task. Experimental results show that GHS provides a satisfactory solution for performance prediction and task scheduling of large applications and has a real potential.

Keywords: Performance prediction and measurement; Task scheduling; Resources sharing; Grid computing; Performance modeling

1. Introduction

With the advance of cyberspace, resource sharing has become a commodity. Many geographically distributed systems, such as Condor [37], NetSolve [12] and Nimrod [1], have been constructed in recent years to support resource and service sharing. The success of these systems inspired and facilitated the formation of a national scale distributed environment, the information power Grid [23]. While noticeable progress has been made in standardization to enable service orchestration and resource collaboration [25,22], Grid computing is still in its infancy. Many research issues remain open, which include how to deliver the diverse autonomous computing power of a Grid effectively to users.

Grid Computing introduces a great challenge in task scheduling: how to partition and schedule tasks in a large, available but shared, heterogeneous computing system. Conventional parallel processing scheduling methods cannot apply to a Grid environment directly where computing resources are shared [24] and non-controlled local jobs co-exist with Grid tasks. A key of Grid task scheduling, therefore, is to estimate the availability of computing resources and to find its influence on the application performance. Some Grid tools have been developed to meet the need. However, these tools are for short-term resource availability. For instance, the well-known network weather service (NWS) [11,47] system only predicts the availability of a computing or communication resource for the next 5 min, and its satisfactory prediction range in general is much less than the 5-min upper bound. Another approach is to avoid the hard issue of predicting resource availability, and use reservation to reserve a computing resource [26]. This approach asks resource owners to give up their privilege and suffers in system utilization. It is useful for high-priority tasks or to show the potential of Grid computing, but has difficulties to be employed in a general enterprise environment. Also, the reservation approach will be more effective and trustworthy if it is based on resource availability prediction.

In this study, we present the design and prototype implementation of a long-term, application-level performance prediction and task scheduling system, namely grid harvest service (GHS) system, for Grid computing. Here long-term means an application requiring hours or more, in contrast to minutes, of sequential execution time; and application-level performance means that the performance is measured in application turnaround time, in contrast to resource availability. GHS consists of the components of performance monitor, performance prediction, task partition and scheduling, and user interface.

* Corresponding author. Fax: +1 312 567 5067.
E-mail address: sun@iit.edu (X.-H. Sun).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.05.008

The prediction component is based on a new performance model [29,42], which is derived from a combination of stochastic analysis and direct-numerical simulation. Unlike other stochastic models, this model individually identifies the effects of machine utilization, computing power, local job service, and task allocation on the completion time of a parallel application. It is theoretically sound and practically feasible.

Utilizing performance prediction, various partition and scheduling algorithms are developed and adopted in the partition and scheduling component for a single sequential task, a single parallel task with a given number of subtasks, optimal parallel processing, as well as a meta-task composed of a group of independent tasks. A heuristic task-scheduling algorithm is proposed to find an acceptable solution with a reasonable cost. GHS uses an adaptive measurement methodology to monitor resource usage pattern, where the measurement frequency is dynamically updated according to the previous measurement history. This method reduces monitoring overhead considerably.

A prototype GHS has been developed for computation intensive applications. Experimental testing has been conducted on Grid nodes at Argonne National Laboratory, Oak Ridge National Laboratory, and in the DOT Grid Testbed using both synthetic programs and a real Grid application, Cactus. Experimental results show GHS outperforms NWS [11,47] and AppLeS [9,10,13,14] in performance evaluation and task scheduling, respectively, for long-term applications. It has a real potential for Grid computing.

The remainder of this paper is organized as follows: Section 2 describes the related work. Section 3 presents the structure and primary components of GHS. The performance evaluation approaches are introduced in Section 4. Performance measurement mechanisms are discussed in Section 5. Section 6 gives task partition algorithms and scheduling schemes. Experimental results are presented in Section 7. Finally, the conclusion and summary are given in Section 8.

2. Related work

Early work in performance evaluation was mostly focused on dedicated systems. The study of usage patterns of non-dedicated workstations is relatively recent [6]. Mutka and Livny [39] reported that the distribution of available time intervals on workstations could be characterized as a combination of several hyper-exponential distributions. Harchol-Balter and Downey claimed that the median remaining life of a process is equal to its current age [30]. These works are observational in nature. Kleinrock and Korfhage [33] gave an analytical expression for the mean and standard deviation of the task completion time based on Brownian motion, which assumes that tasks are assigned to loaded workstations. Further, the effects of different factors cannot be analyzed due to the inherent limitation of their method. Leutenegger and Sun [35] put forward an analytical performance model to investigate the effect of a remote task on local jobs of the workstation owner and vice versa. An effective prediction equation was derived for homogeneous non-dedicated systems. Most recently, Gong et al. have introduced a more general model for heterogeneous non-dedicated network computing [29]. Their model was derived from a combination of rigorous mathematical analysis and intensive simulation.

There are several on-going projects on performance evaluation in parallel or distributed programming environments. Paradyn parallel performance tools [38] is a known performance evaluation system. The technical features of Paradyn are dynamic instrumentation, W3 (why, when, and where) search model and uniform data abstraction. Paradyn measures the performance of an application. However, it does not provide performance analysis and prediction based on resource usage pattern. Tuning and analysis utilities (TAU) [41] was developed at the University of Oregon. Its salient features are instrumentation at the source code level, message trace and visualization, standard format for performance files, and further analysis based on the recompilation and rerun of the application with different profile statistics options of libraries. It is a post-execution performance analysis system. The Prophesy project is centered on the interaction or coupling between kernels with an application [43]. Another related work is SCALEA [44], which provides the performance analysis for individual code regions of applications. These systems focus on application performance in a dedicated parallel system instead of a non-dedicated distributed environment.

Network weather service (NWS) [11,47] monitors and forecasts resource performance on-line. It provides system performance sensors, various simple forecasting methods, dynamic predictor selection and web-based visualization interfaces. Resource prediction system (RPS) Toolkit [16] predicts the CPU availability of a Unix system over a small time range with time series techniques. These works are for non-dedicated environments. However, they only predict the short-term availability of non-dedicated resources. There is no application-level performance analysis and long-term prediction.

Most scheduling methodologies in distributed systems consider application performance or system load balance [8,32]. They are based on either current system usage or advanced resource reservation mechanisms. Condor system provides a matchmaking mechanism to allocate resources with ClassAds [40]. The scheduling strategy is based on the mapping of the users' ClassAds. It ranks available machines according to their latest usage, instead of application-level performance prediction. Grid advance reservation API (GASA) [26] is a subsystem of Globus project. It provides resource reservation mechanisms so that applications can receive a certain level of service. Legion system also supports resource reservation [36]. These reservation approaches try to have dedicated resources in a non-dedicated environment, which may or may not be feasible in an actual engineering environment. Some other projects, such as Nimrod, consider the economy issue. In our study, we focus on the application performance in task scheduling. The experience in the development of the GrADS project [9] and other Grid projects has demonstrated that the integration of performance evaluation mechanisms with application is pivotal to the success of Grid computing [22]. However, there is no existing performance prediction and task scheduling system designed for a large scale-application in

a shared Grid environment. NWS is designed for short-term resource predictions. The AppLeS Grid task scheduling system [10,11] uses the short-term resource prediction provided by NWS for task scheduling and only focuses on independent meta-tasks. GHS is designed to cover the non-existence and is a complement of NWS and AppLeS.

3. The design of GHS system

Appropriated scheduling of diverse, autonomous network resource is an essential of Grid computing. The goal of GHS is to minimize the application execution time. There are many factors affecting the design considerations, such as scheduling cost, application types (e.g. parallel program or meta-task) and the workload of applications (e.g. long-term or short-term). Different emphasis on these factors may lead to various scheduling strategies. Whatever factors are considered, however, task scheduling of a distributed application generally includes several basic steps: resource selection, task partition, performance evaluation, and finally scheduling. Resource selection is to choose a subset of the available resources. Task partition can be either partitioning or grouping. Partitioning divides a parallel application into subtasks and then assigns each subtask to a resource in this subset. Grouping clusters subtasks in a meta-task and then assigns each set of subtasks to a resource. A meta-task is composed of a set of independent indivisible subtasks. Strategies of resource selection and task partition can be based on resource availability information. Performance evaluation is to estimate the application execution time for a given task partition plan using a performance model. A straightforward approach of scheduling is to examine the performance of all possible combinations of assignments. An optimal scheduling plan then can be selected accordingly based on the predicted application performance. A general flowchart of scheduling is shown in Fig. 1. GHS follows the general scheduling frame. The difference in GHS is that the performance evaluation and scheduling has to consider the variation of resource availability. Resource availability has patterns and can be predicted for long-term applications. For a given resource, however, the availability pattern may vary over time. To cope with the variation, GHS also has an additional rescheduling functionality. When a major pattern change is identified, GHS will reschedule tasks accordingly.

The strategy used in each step depends on the application specific information and the system specific information. For example, based on the ensemble size of the Grid and scheduling cost, different resource selection mechanisms can be chosen in Step 1. If the number of available resources is small, an exhaustively resource selection method is usually applied to find an optimal solution. If we have a large number of available resources, a heuristic algorithm should be applied for a near-optimal solution. The design of GHS is component-oriented and follows the basic scheduling principle. GHS can be easily integrated into other scheduling systems and vice versa. Fig. 2 depicts the primary components of GHS (shaded areas) and their relations with other general Grid services in a Grid environment provided by the Globus middleware [28].

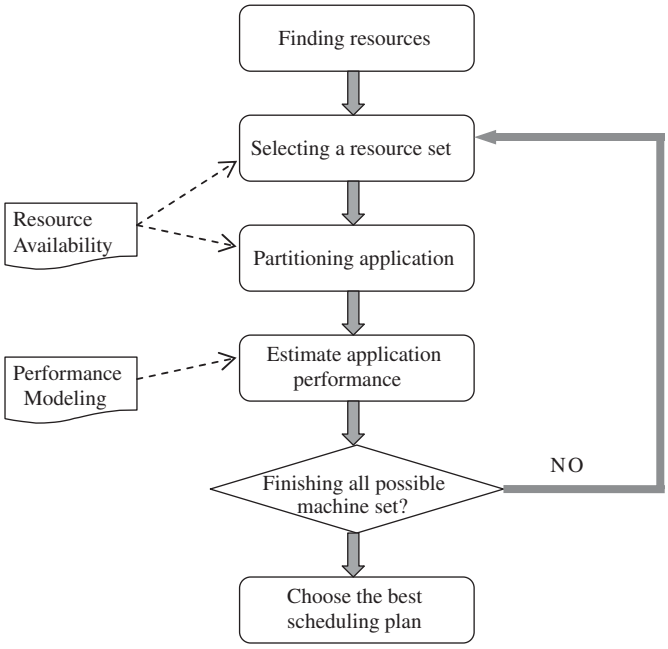

Fig. 1. Flowchart of scheduling a distributed application.

These components are:

- Application-level predictor: This component estimates the application performance based on the map information provided by the Task Allocator component and the system information provided by the System-level Predictor component. It identifies the distribution of an application execution time in a shared environment.
- System-level predictor: The System-level Predictor analyzes resource usage patterns to provide an estimation of system performance in a future period. The input of this component is the preprocessed system parameters stored in a performance database. The structure of the database could be either distributed or centralized depending on the system scale.
- Task allocator: This component decides how to partition a parallel program or how to group subtasks of a meta-task. The input of this component is the application characteristics (workload, application type, dependency of subtasks of an application). The output is a mapping of subtasks on a given set of computing resources.
- Task scheduler: The Task Scheduler determines a scheduling plan for a large-scale application to provide an optimal or near-optimal solution for its running in a shared environment. It is actually a searching process. Users submit a parallel application or a meta-task for task scheduling. The task scheduler checks potential available resources in the system and then searches for the best set of resources to assign the application.
- Performance data management: This component implements data filtering and reduction techniques for extrapolating system and application performance information on each resource. The raw performance data is collected through different types of sensors, such as CPU sensor, I/O sensor, and network sensor.

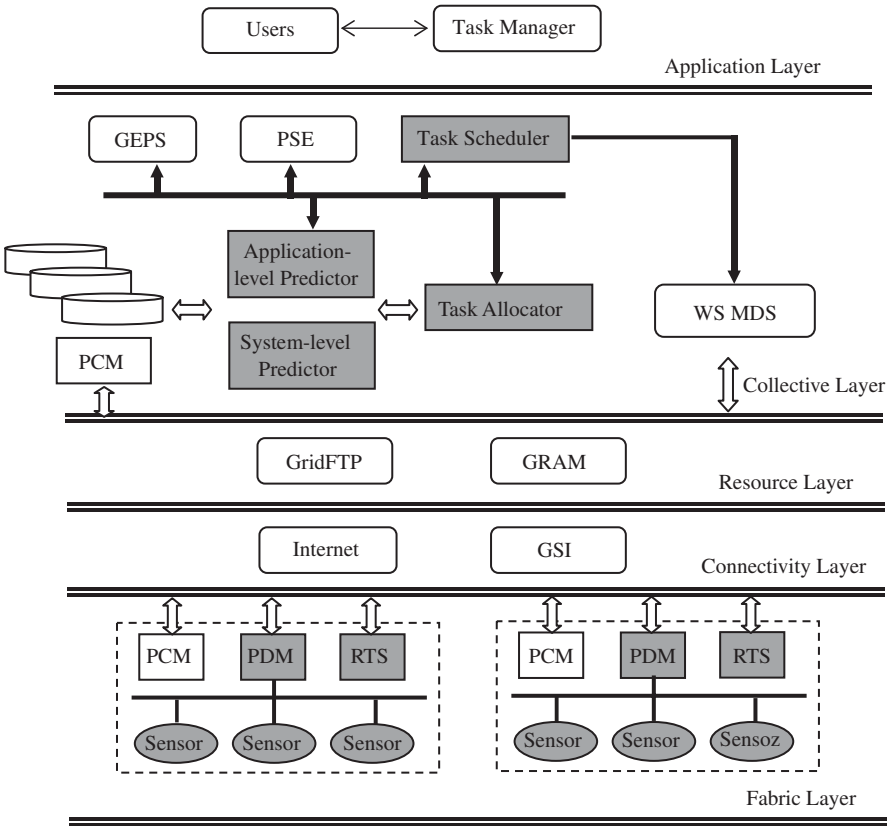

Fig. 2. Framework of GHS performance evaluation and task scheduling system.

- Reschedule trigger system: This component analyzes the collected application and resource performance data to detect whether resources present abnormal performance from their historical records. If an abnormal resource behavior is identified, application will be rescheduled to prevent potential performance loss. This component is included in GHS to enable reliable scheduling in shared environments, like a Grid.

The task manager, which is responsible for task management, is located in the Application layer. It sends a request to the scheduler component in the collective layer for resource allocation. The scheduler component contacts the index service provided by WS monitoring and discovery system of Grid to locate potential available resources. The prediction component can also serve other Grid service, such as the Grid-enabled programming system (GEPS) and problem solving environment (PSE), in a Grid runtime system. The prediction component accesses the performance database to estimate the task completion time. The Grid performance communication manager (PCM) component is used to collect performance data. The Grid FTP service can be used to handle the transfer of applications and their data files, and the GRAM can be used to dispatch subtasks of applications on resources. In the fabric layer, the GHS performance data manager (PDM) component on each resource is responsible for measuring system and application information by using various sensors and the reschedule trigger system (RTS) [20] analyzes the collected performance data and automatically triggers rescheduling when abnormal situations are detected.

4. Performance evaluation

Performance prediction is a foundation of task scheduling in shared environments [11]. Current scheduling systems, such as AppLeS [10,13,14], PEGASUS [15], CONDOR-G [27], and Stork [34], involve some kind of prediction. However, they do not consider the variation of resource availability on application performance and assume the utilization of each resource is fixed during each execution. Instead of using a fixed utilization for resource availability, we model the resource usage pattern with a novel M/G/1 queue system. This model individually identifies the effects of machine utilization, computing power, local job service, and task allocation on application performance [29].

4.1. Prediction of application execution time

The execution time of an application is affected by application characteristics, system status, and parallel processing. In this section, we introduce the underlying prediction model of the GHS system. To distinguish an application under scheduling with other competing processes, we call the application the remote task and other competing processes local jobs. We assume the arrival of local jobs follows a Poisson distribution with λ. The service time of local jobs follows a general distribution with mean $1/\mu$ and standard deviation σ. These assumptions are based on the observations of machine usage patterns reported by researchers in Wisconsin-Madison et al. [2,6,39]. Since we only consider large-scale applications, we also

assume all the available resources do not have remote tasks. Otherwise, the utilization of resources would be high and would not be available for new tasks.

4.1.1. Computation time of a remote task

In GHS system, a remote task is given a lower priority than local jobs so that the remote task is less intrusive. Let w denote the workload of the remote task and τ denote the computing capacity of a machine which is defined as the amount of workload that can be finished in a unit time. Given a machine is idle when a remote task arrives at the machine, the completion time of the remote task can be expressed as

$$T = X_1 + Y_1 + X_2 + Y_2 + \cdots + X_S + Y_S + Z,$$

where X_i and Y_i $(1 \leqslant i \leqslant S)$ represent the computing time consumed by the remote task and the local jobs, respectively; and S is the number of interruptions due to local job arrival; and Z is the execution time of the last phase of the remote task. By defining

$$U(S) = \begin{cases} 0, & \text{if } S = 0, \\ Y_1 + Y_2 + \cdots + Y_S, & \text{if } S > 0, \end{cases}$$

we can obtain the distribution of the remote task execution time T as [29]

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

To calculate the distribution of the remote task completion time, we need to identify the distribution of $\Pr(U(S))$. Using the well-known result in queuing theory, we can get the mean and variance of the remote task completion time. The mean and variance of $U(S)$ given $S > 0$ are thus calculated, respectively as

$$E(U(S)|S > 0) = \frac{1}{1 - e^{-\lambda w/\tau}} \frac{\rho}{1 - \rho} \frac{w}{\tau}, \quad (2)$$

$$V(U(S)|S > 0) = \frac{1}{1 - e^{-\lambda w/\tau}} \frac{\rho}{(1 - \rho)^3} \frac{(\theta^2 + 1)}{\mu} \frac{w}{\tau}, \quad (3)$$

where $\rho = \lambda/\mu$ is the machine utilization and θ is the coefficient of variation of service. Simulation results indicate that Gamma, Lognormal or Weibull are among the best-fit distributions to describe the $\Pr(U(S_k)$. In GHS, we use the Gamma distribution in the calculation of the distribution of the remote task completion time.

In the above discussion, the remote task is assumed as a sequential job. In a Grid environment, we usually deploy large-scale parallel applications which are often partitioned into independent subtasks for parallel processing. Each subtask is assigned to a different machine. The computation time of a parallel application is thus the maximum of the execution time of its subtasks. After the distribution of the computation time of each single subtask is identified, the cumulative distribution function of a remote parallel task computation time can be calculated as

$$\Pr(T \leqslant t) = \begin{cases} \prod_{k=1}^{q} [e^{-\lambda_k w_k/\tau_k} + (1 - e^{-\lambda_k w_k/\tau_k})\Pr(U(S_k) \leqslant t - w_k/\tau_k | S_k > 0)] & \text{if } t \geqslant w_{\max}, \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

where $\rho_k = \lambda_k/\mu_k$ is the machine utilization, w_k is the subtask workload on machine m_k $(1 \leqslant k \leqslant q)$, and $w_{\max} = \text{Max}\{w_k/\tau_k\}$.

By evaluating the mean and coefficient of the application completion time on a group of available machines with Eq. (1) or Eq. (4), the remote task to be scheduled can be assigned to the most appropriate resources. In this study, we focus on task scheduling of computation-intensive applications. We assume communication cost is either negligible small compared to computing or is a constant which can be considered as a part of the workload. We do not consider the variation of network bandwidth availability. Our model cannot extend to communication networks due to the complicated short and long-range temporal dependence characteristic of heterogeneous network traffic and the underlying routing policies. Also, no existing network model can provide a satisfactory solution for a long-term network performance estimate. Developing a practical long-term network performance prediction mechanism is a challenging research issue facing the community today. However, if such a network performance model were developed, GHS would be extendable to communication extensive applications.

4.2. Prediction of resource availability

From the above discussion, we can see that the application execution time in a shared environment is determined by the application workload (w) and resource parameters (λ, ρ, σ, τ). To predict the distribution of the remote task completion time, we first need to determine these parameters. While the application workload and the resource speed are static values and can be easily obtained by analyzing the application structure (or provided by users) and running the application benchmark respectively, the system parameters λ, ρ, and σ may vary with time. For example, the job arrival rate λ could be less at night. In GHS, the system-level predictor determines these system parameters via measurement and prediction.

In GHS, we first aggregate the original measurement time series into an interval time series. Then we choose the mean-based method, which uses the arithmetic average of the aggregated values as the estimate of the system parameter value over the time interval. Similar methods are used in NWS [46] and RPS [16]. The difference lies in the selection of the sample space. NWS and RPS use the aggregated values over a

Objective: prediction of a system parameter
Input: system parameter η, time interval
Output: estimated value of η in the coming time interval

Begin
Aggregate the original measurement time series to generate the past-sample-space, each sample is an aggregated value, which is defined as $\frac{1}{|X|}\sum_{i=1}^{|X|} v_i$ where X is the set of v_i measured during the time interval.
Calculate V_p;
Aggregate the original measurement time series to generate the day-sample-space;
Calculate V_d;
Aggregate the original measurement time series to generate the week-sample-space;
Calculate V_w;
Output $\alpha^* V_p + \beta^* V_d + \gamma^* V_w$ while $(\alpha+\beta+\gamma)=1$ and $\alpha, \beta, \gamma > 0$
End

Fig. 3. System parameter prediction algorithm.

number of continuous time intervals as the sample space and then use the average to predict the value in the next time interval. We call this kind of sample space past-sample-space. Past-sample-space may work for short-term estimations (in terms of seconds or minutes). However, as indicated in [45], long-term resource behaviors often present seasonal patterns. Considering these seasonal effects, we add two different sample spaces: day-sample-space and week-sample-space. The day-sample-space includes the aggregated values over a number of the same time intervals in a day, and the week-sample-space includes the aggregated values over a number of the same time intervals in a week. We define V_{p}, V_{d} as the arithmetic average over past-sample-space, V_{d} as the arithmetic average over day-sample-space, and V_{w} as the arithmetic average over week-sample-space. After calculating V_{p}, V_{d}, and V_{w}, we use $\alpha^* V_{\mathrm{p}} + \beta^* V_{\mathrm{d}} + \gamma^* V_{\mathrm{w}}$ to estimate the value over the next time interval where $(\alpha + \beta + \gamma) = 1$. The system parameter prediction algorithm is illustrated in Fig. 3. In the current implementation, we set $\alpha = \beta = \gamma = \frac{1}{3}$. A gradient-descent strategy is applied in NWS prediction to dynamically set the size of a "sliding window" [46]. Similar methods can be utilized to adjust α, β, γ so they can adapt to the system change. Other forecast models, such as LAST model, AR model, and ARIMA model [16], can be used to replace the mean-based method in this algorithm.

5. Performance measurement

Four system parameters, λ, ρ, σ and τ, are used in our prediction model. λ is the local job arrival rate, ρ is the machine utilization, σ is the standard deviation of service time, and τ is the computing capacity for a given machine. τ is a constant and can be obtained by running application benchmarks. We focus on the measurement of λ, ρ and σ.

Supposing parameter x over a time interval has a population with a mean and a standard deviation and we have a sample $\{x_1, x_2, \ldots, x_n\}$, the smallest sample size with a desired confidence interval and a required accuracy r is given by $n_s = (100 z_{1-\alpha/2} d / r\bar{x})^2$ [31]. The desired accuracy of r percent means that the confidence interval is $(\bar{x}(1 - r/100, \bar{x}(1 + r/100))$. If the confidence interval is 95% and accuracy is 5, we get

$$n_s = 1536.64 \left(\frac{d}{\bar{x}}\right)^2, \tag{5}$$

where the sample mean is $\bar{x} = (1/n)\sum_{i=1}^{n} x_i$ and the sample standard deviation is

$$d = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}.$$

We assume that parameter x is a random variable with a fixed mean and a fixed standard deviation during a continuous 24-h period. Parameter x is measured in n_s time intervals during 24 h. At the end of each hour, we calculate n_s with $\bar{x}$ and d over the previous 24 h using Eq. (5). In this way, we dynamically adjust the number of time intervals to adapt the possible variance of x. A number $(n_s/24)$ of x_i will be measured for the next hour.

The Unix utility *vmstat* is used to measure ρ. It accesses performance statistical data, which is collected and maintained by the kernel system. *vmstat* directly outputs the resource utilization. The Unix utilities *ps* and *lastcomm* are used to obtain local jobs' execution information at the beginning and the ending of T_{interval} to calculate λ and σ. *ps* shows the process information of the active job while *lastcomm* presents the processes information of the finished jobs. The detailed measurement methods are given in [42].

In GHS, the resource performance data collection is provided by the Performance Data Management component. We calculate the number of measurements for the next hour according to the system history over the previous 24 h. This method can dynamically adjust the measurement number to reduce the measurement cost. The GHS measurement consumes very little CPU resource (less than 1%).

Assumption: a meta-task composed of a number of independent, indivisible subtasks, $S_T = \{t_1, t_2, ..., t_p\}$ and a list of machines $\{m_1, m_2, ..., m_q\}$.

Objective: find a task group $G_k = \{t_{k_1}, t_{k_2}, ..., t_{k_n}\}$ for machine m_k $(1 \le k \le q)$

Begin

/* C_k denotes the current estimated completion time of the assigned subtasks on machine m_k */

$C_k = 0$, $G_k = \phi$, $(1 \le k \le q)$; $i = 1$;

While $S_T \neq \phi$

  For each $t_i \in S_T$ **Do**

    $j = 1$;

    While $j < q$

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

      /* $E(T_{i,j})$ is the expected execution time of task t_i on machine m_j */

      $j = j + 1$;

    End While

    Find machine k where $C_k + E(T_{i,k})$ is minimal;

  End For

  Find a map of (t_u, m_v) where $C_v + E(T_{u,v})$ is minimal;

  $G_v = G_v \cup \{T_u\}$; $C_k = C_k + E(T_{u,v})$; $S_T = S_T - \{T_u\}$; $i = i + 1$;

End While

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

End

Fig. 4. Min–min task group allocation algorithm.

6. Task scheduling

Task scheduling in a shared environment involves the integration of application specific and system specific information. GHS is designed to support task scheduling based on the performance estimation provided by the application-level predictor and the system-level predictor to minimize the make-span of applications. We have designed different task scheduling algorithms for various classes of applications.

6.1. Task allocation

Two classes of distributed applications are investigated. One is parallel application, which can be partitioned arbitrarily into subtasks. Another is meta-task, which is composed of independent indivisible subtasks. A typical example of meta-tasks is the parameter sweep application, a widely used Grid application [13].

For task scheduling of distributed applications, the first problem is how to partition an application and allocate subtasks to machines so that we can achieve an optimal performance for a given number of machines. After identifying the subtask demand w_k on each machine m_k, we can use Eq. (4) to calculate the mean and variance of the distributed application completion time. A natural optimal partition strategy is to assign each machine a certain amount of workload so that subtasks on different machines are finished at the same time. We call this mean-time allocation. For a parallel application, supposing that the mean subtask completion time is α, we can get the subtask demand $w_k = \alpha(1 - \rho_k)\tau_k$. Since $W = \sum_{k=1}^{q} w_k$, the subtask workload can be expressed as

$$w_k = \frac{W}{\sum_{k=1}^{q} (1 - \rho_k)\tau_k}(1 - \rho_k)\tau_k. \tag{6}$$

By comparing the parallel application completion time on different sets of machines with Eq. (4), we can identify the best machine set for running this application.

For a meta-task, we may not be able to make the assigned task on each machine complete at the same time because the subtasks cannot divide arbitrarily. So we cannot use Eq. (6) to calculate the subtasks' workload on each machine directly. We apply a min–min heuristic algorithm to group indivisible subtasks and map each subtask group to one of the resources based on the estimation of execution time. In this algorithm, we assume that an application is composed of a number of independent tasks, $\{t_1, t_2, \ldots, t_p\}$ with workload $\{w_1, w_2, \ldots, w_p\}$ and we have a list of machines $\{m_1, m_2, \ldots, m_q\}$, Fig. 4 shows the min–min task group allocation algorithm. The intuition behind this algorithm is to assign subtasks of a meta-task to resources one by one while keeping the difference among the subtask completion time on each machine as little as possible. In GHS, the task allocation of parallel applications is performed by the task allocator.

Objective: scheduling a sequential task on a set of machines

Begin

List a set of idle machines that are lightly loaded over an observed time period;

$M = \{m_1, m_2, \ldots m_q\}$;

$p' = 1$;

For each machine m_k $(1 \le k \le q)$,

Use Eq. (1) to calculate the mean of the remote task execution time, $E(T_k)$;

Use Eq. (1) to calculate the coefficient of variation of the remote task execution time, $Coe.(T_k)$;

If $E(T_{p'})(1 + Coe.(T_{p'})) > E(T_k)(1 + Coe.(T_k))$, then $p' = k$;

End For

Assign task to the machine $m_{p'}$;

End

Fig. 5. Sequential task scheduling algorithm.

Assumption: solving a parallel task on p machines. Each machine only hosts one subtask.
Objective: Scheduling a parallel task on a given number of machines

Begin

List a set of idle machines that are lightly loaded over an observed time period,

$M = \{m_1, m_2, \ldots m_q\}$;

List all the possible sets of machines, $S = \{S_1, S_2, \ldots S_z\}$, $S_i \subset M$ and $|S_i| = p$;

$p' = 1$;

For each machine set S_k $(1 \le k \le z)$,

Use Eq. (6) to partition subtasks to each machine in S_k;

Use Eq. (4) to calculate the mean and coefficient of variation of the remote task completion time, $E(T_{S_k})(1 + Coe.(T_{S_k}))$, approximating $\Pr(U(S_k) \le u \mid S_k > 0)$ with the Gamma distribution;

If $E(T_{S_{p'}})(1 + Coe.(T_{S_{p'}})) > E(T_{S_k})(1 + Coe.(T_{S_k}))$, then $p' = k$;

End For

Assign the parallel task to the machine set $S_{p'}$;

End

Fig. 6. Parallel task scheduling algorithm with a given number of subtasks.

6.2. Task scheduling

The task scheduler is responsible for finding an appropriate machine set for an application. It is supported by the task allocator and the application-level predictor. The task scheduler supports different scheduling algorithms for different application requirements. Fig. 5 gives the scheduling algorithm for a single sequential task. Based on the estimate of the expectation and variance of the application completion time, the algorithm finds the best machine for a task. Fig. 6 shows the scheduling algorithm for a single parallel application with a given degree of parallelism (under our assumption of one subtask for one machine, the degree of parallelism equals the number of machines or subtasks used for the parallel processing). Let q denote the number of available machine and p denote the application's subtask number. We need to go through C_q^p possible machine sets to get an optimal task scheduling decision. We also develop optimal parallel processing where the application can be partitioned arbitrarily into any number of subtasks. To achieve an optimal scheduling plan, we have to search 2^q possible degree of parallelisms and machine combinations. The cost is quite high when the machine set is large. A heuristic task-scheduling algorithm, as shown in Fig. 7, is proposed to find a near optimal solution with a reasonable cost. It includes two basic steps. The first step of this algorithm is to sort each lightly loaded machine according to $(1 - \rho_k)\tau_k$. Since ρ_k is the machine utilization, $(1 - \rho_k)$ means how much percent of the machine's CPU resource is available for a remote task. τ_k is the machine's computing power. So the production of $(1 - \rho_k)$ and τ_k stands for the amount of the machine's computing power available to the remote task. A higher value $(1 - \rho_k)\tau_k$ of a machine indicates this machine has more available computing power for an application and thus should be considered first. The second step of this algorithm is to use the bi-section search to find the local optimal based on the ordering. In Fig. 7, w is the application work demand, μ' is the average of local jobs' work demands. Leutenegger and Sun [35] show that the task ratio,

Assumption: a remote task can be partitioned into any size of subtasks. Each subtask will be assigned to a machine respectively.
Objective: scheduling a remote task heuristically to reach a semi-optimal performance

Begin
List a set of idle machines that are lightly loaded over an observed time period, $M = \{m_1, m_2, \ldots m_q\}$;
Sort the list of idle machines in a decreasing order with $(1-\rho_k)\tau_k$, $M' = \{c_1, c_2, \ldots c_q\}$;
$a = 1$, $b = \min\{|M'|, \frac{w}{4 * 1/\mu'}\}$;
Repeat
$c = \lfloor (a+b)/2 \rfloor$
/* $f(x)$ denotes $E(T_{C(x)})(1 + Coe.(T_{C(x)}))$ where $C(x) = \{c_1, c_2, \ldots c_x\}$ */
If $f(a) = \min\{f(a), f(b), f(c)\}$ **then** $b = c$
Else If $f(b) = \min\{f(a), f(b), f(c)\}$ **then** $a = c$
Else If $f(c) < f(c+1)$ **then** $b = c$
Else $a = c$
Until $a + 1 = b$
If $f(a) < f(b)$ **then**
Assign parallel task to the machine set $C(a)$;
Else Assign parallel task to the machine set $C(b)$;
End

Fig. 7. Heuristic task scheduling algorithm.

the ratio of the remote task work demand to the mean demand of machine's local jobs, should be large enough to achieve acceptable efficiency. Here we choose it to be at least 4. Note that in Figs. 6 and 7, we use the mean-time method to allocate subtasks to each machine. When the allocation strategy is the min–min task group allocation methodology, these task-scheduling algorithms could also be applied for meta-task scheduling. A typical example of meta-tasks is the parameter sweep application [13]. AppLeS [13,14] has investigated how to schedule the parameter sweep application in a Grid computation environment. However, its scheduling algorithms are based on minimizing each individual task completion time, where minimizing each task does not necessarily lead to an optimized completion time of the whole application.

A self-adaptive task scheduling algorithm is designed and implemented to deal with a situation when resources present abnormal performance from their historical records, which may be due to malicious users/application behaviors or a sudden change of local job's work demand. A self-adaptive task scheduling algorithm has been proposed to decide whether we should reschedule the application and which resource we should choose for task reallocation [48].

7. Experimental results

We have developed a prototype GHS system and conducted experimental testing on machines at the Argonne and Oak Ridge national laboratories, as well as in the DOT Grid testbed [18]. The performance of GHS prediction is first examined to verify its accuracy and feasibility. Then we evaluate three task allocation methods to examine GHS task partition strategy. The completion time of a remote task over different numbers of machines with different scheduling methods (optimal, heuristic, random) is also compared. After that, we compare the AppLeS schedule with the GHS schedule. Finally, we test the efficiency of our dynamic measuring methodology in reducing the measurement cost.

7.1. GHS system-level and application-level prediction

The estimation of application performance in a shared environment is based on the prediction of resource availability, which is provided by system-level prediction. In this subsection, we investigate the prediction error of GHS both at the system level and the application level. We first examine the prediction error of GHS for a sequential task. We then test GHS prediction performance with a synthetic parallel task in both a simulated distributed environment and an actual Grid productive machine. Finally, we evaluate GHS prediction accuracy for a real application in a Grid Testbed.

Two performance metrics are generally used in the literature to evaluate the accuracy of a prediction model. One is percentage prediction error, which is defined as $|(Prediction - Measurement)/Measurement|$[46]. Another is square prediction error, which is defined as $(Prediction - Measurement)^2$ [17,46]. For large-scale applications, the square prediction error could be very big even for excellent prediction. This is especially true for scalable computing, where the square prediction error may increase with the problem size. Percentage prediction error is a more appropriate metric for large-scale applications and for scalable computing. In our experiments, we are mainly concerned about the effectiveness of GHS prediction for large applications with different sizes. Thus, we choose the percentage

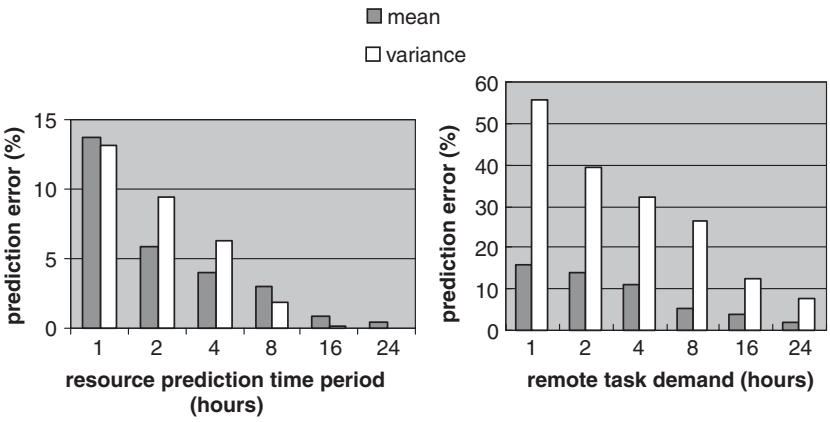

Fig. 8. Mean and variance of prediction error of utilization and remote task completion time on a single machine.

prediction error and simply call it prediction error throughout the study.

An experiment is first conducted to test the prediction error of GHS for a sequential task. Fig. 8 shows the mean and variance of the prediction error of resource utilization and completion time of a remote task with different workloads (from 1 to 24 h) on a Sun workstation. The remote task is a replication of NAS Benchmarks (BT, CG, LU, MG, IS and SP) [7]. The class type of these benchmarks is "A" or "W". The local job's lifetime is simulated with $2.0/x$ [5], which follows the observation of real-life processes in [30]. x is a random number between 0 and 1. We also simulate the machine usage pattern based on observations from SDSC Paragon logs and CTC SP2 logs [19]. The experimental results show, with the increase of remote task workload, the mean and variance of the prediction error get smaller. The left graph in Fig. 8 gives the mean and variance of the prediction error of resource utilization and the right one gives the mean and variance of the prediction error of remote task completion time. We can find that the prediction error of remote task completion time is larger than that of system utilization. This may be explained by the real system time-sharing scheduling strategy. In Unix system, even when the priority of a remote task is set to the lowest, it still can occupy a small part of CPU resource. Thus the completion time of a short-term remote task is usually smaller than the analytical result. With the increase of remote task workload, the impact of this scheduling strategy on the prediction error gets smaller. When a remote task's workload is more than 8 h, the mean of the predication error of task execution time is less than 10%.

We then investigate the prediction error of a parallel task. As we focus on computation intensive applications, we use a CPU-bound multi-level loop program for testing. The resource usage pattern of local jobs is set the same as sequential processing testing. We use a Sun ComputeFarm, named Sunwulf, at Illinois Institute of Technology (IIT) to simulate a general distributed environment in which each machine has its own job arrival rate and service rate. The setting of different resource usage patterns on each machine is based on observation of real-life processes [30,19] and is used for all simulated Grid environments throughout this study unless indicated explicitly. We conduct our experiment six times. Fig. 9 gives the mean and variance of the prediction error on the parallel task completion time with different task work demands (from 32 to 512 h) on 32 nodes of Sunwulf. Notice that the task demand is the total work demand. For example, with a 32-h task work demand, the average workload on each of the 32 processors is 1 h. As expected, a larger task workload leads to a smaller variance. The mean and variance of the prediction error get smaller with increase in task work demand.

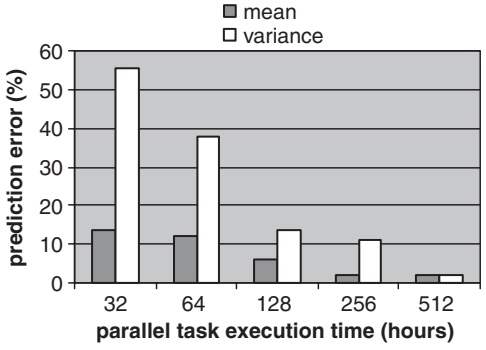

Fig. 9. Mean and variance of prediction error of parallel task completion time with different total task demands.

We have also evaluated our prediction model for computation cost on an actual Grid environment. Fig. 10 shows the mean and variance of the prediction error of the parallel task completion time on *Pitcairn*, a productive machine at Argonne National Laboratory. *Pitcairn* is a multiprocessor with eight 250 MHz UltrasparcII processors and 1 GB of shared memory. It is a Grid node shared by many users. The result again shows that the mean and variance of the prediction error get smaller as the demand of a remote task increases. When the demand of the remote parallel task is 8 h, a 1-h workload is assigned to each subtask. The mean of the prediction error is about 9.31%. The prediction error with 16-h remote task demand is about 4.18%. Also we find that the prediction error is reduced more quickly than that on a single workstation. This is due to the property of probability modeling: with more processors and more samples, the predicted results are more accurate.

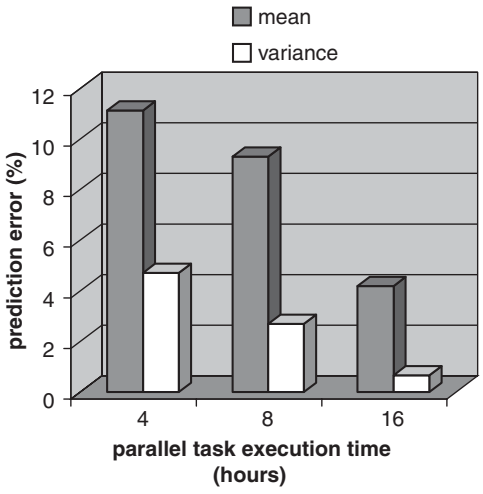

Fig. 10. Mean and variance of prediction error of parallel task completion time on a multiple-CPU machine.

Fig. 11. Mean and variance of prediction error of Cactus application execution time on multiple clusters.

To evaluate GHS prediction accuracy for a real application in an actual Grid environment, we examine the prediction error of the application completion time of Cactus in the DOT Grid Testbed [18]. Cactus [3,4] is a numerical simulation of a 3D scalar field produced by two orbiting astrophysical sources. In our experiments, we use the parallel version of Cactus application. It decomposes the 3D scalar field into sub-fields, and each sub-field is mapped onto a processor. An overlap region is placed on each processor. We use the iteration number to adjust the application execution time. The DOT interconnects computer clusters at the Argonne National Laboratory, National Center for Supercomputer Applications, Northwestern University, University of Chicago, University of Illinois at Chicago, as well as IIT via the dedicated, high-speed "I-Wire" network. Each cluster is composed of one server and multiple computing nodes. In the experiments, we use one server and one node from the IIT cluster, three nodes from the ANL cluster, and three nodes from the UC cluster. Local jobs are simulated with the same method as before. Fig. 11 gives the mean and variance of the prediction error on the Cactus parallel application completion time with different task work demands (from 8 h to 128 h) on the eight nodes (from three clusters) of the DOT Testbed. Compared with the experimental results shown in Fig. 9, we find that the prediction error of the Cactus application completion time is usually larger than that of the synthetic application for the same workload per node. This may be due to the Grid communication and middleware cost. Cactus is an iterative program. At the end of each iteration, data in the overlap area needs to be exchanged between neighboring nodes. In our experiment, we do not calculate the Cactus communication cost separately. Instead, we consider it as a part of application workload. We measure the whole application execution time several times in a dedicated distributed environment and use the average as the workload for the prediction of application completion time in a corresponding shared environment. This method includes communication cost as workload but may introduce some prediction error when the underlying network performance varies during application running. However, we observe that GHS prediction still provides satisfactory predictions, especially for large applications. When the Cactus execution time has an average of 8 h per node (64 h sequential workload), the predictor error goes down to 5.9%.

7.2. GHS task partition and scheduling

Task scheduling in GHS is performed by the collaboration of the system-level predictor, the application-level predictor, the task allocator and the task scheduler. In the above experiments, we have demonstrated the efficiency of GHS prediction modeling. In this subsection, we investigate the performance of both task allocation mechanisms and task scheduling algorithms of GHS.

GHS uses the mean-time allocation to partition the workload of a parallel application. In our experiment, we compare the performance of the mean-time allocation with two other allocation approaches. One is the equal-load allocation, where the remote task workload is divided into equal sub-workloads and then assigned to each machine. Another is the heterogeneous equal-load allocation, which allocates each machine the sub-workload matching its theoretical computing power.

We test the efficiency of these allocation approaches in two workstations. Workstation Scala has an average utilization of 50% while workstation Macro has an average utilization of 20%. The machines have a speed ratio of 1.33:1. Fig. 12 shows the remote task completion time with these three allocation approaches on the two machines. The parallel task demand increases from 1 h to 8 h. Results show that the mean-time allocation is the best. The time saved by the mean-time allocation algorithm is 20–25% for large tasks. This difference is significant. We further conduct our experiment on 64 nodes of Sunwulf with different parallel workloads (16 h and 32 h). Because each of the 64 nodes used in our test has the same computing capacity, we compare only the equal-load and mean-time

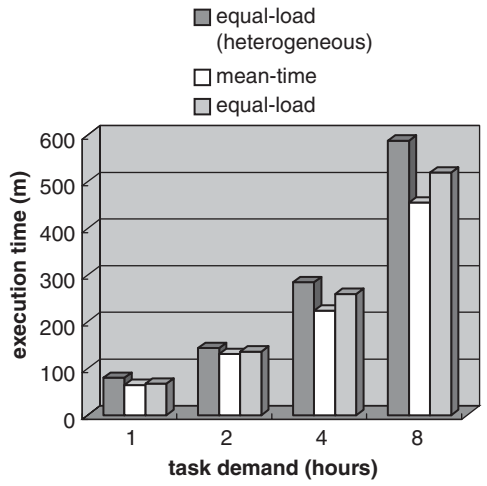

Fig. 12. Comparison of three partition approaches.

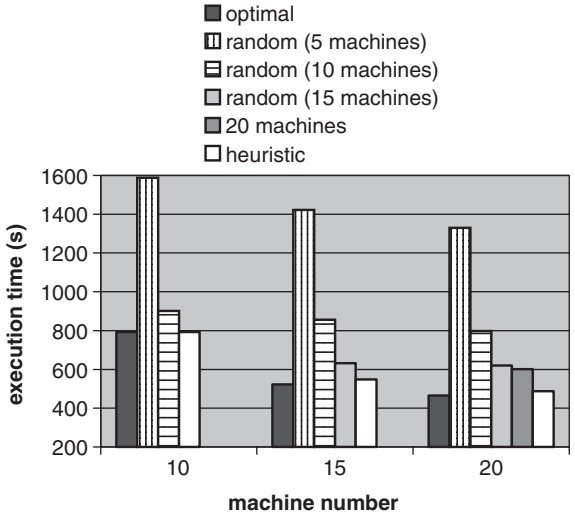

Fig. 13. Average execution time with different scheduling strategies.

approaches. This experimental result shows that the mean-time allocation is better than the equal-load. Around 25–40% of time is saved.

The task allocation algorithm is used to identify how much workload of a remote task should be assigned to each machine in a given set of machines while the task scheduling algorithm aims to find the best set of machines from a list of available machines. Task allocation is another factor distinguishing GHS from other existing Grid scheduling systems where allocation is either not considered or equal-load allocation is used. In Section 6.2, we have discussed various scheduling algorithms for four cases. A heuristic task scheduling algorithm is proposed because of the high computing cost of optimal task scheduling.

We have conducted experiments to compare the performance of two scheduling algorithms, the optimal and heuristic task scheduling on Sunwulf. The experiment is executed 10 times over different number of nodes, 10, 15 and 20. In each case, besides the optimal and heuristic task scheduling methods, we also select a subset of machines for task allocation at random. The average completion time of a remote task with different scheduling algorithms are given in Fig. 13. The experimental results show that the execution time of the remote task and the number of utilized machines of heuristic task scheduling are close to those of optimal task scheduling while the heuristic scheduling cost is far less than the optimal scheduling cost. When scheduling task among 20 available machines, 14 machines are identified for optimal scheduling and 13 machines are used for heuristic scheduling. The average run time is 464.9 s for optimal scheduling and 486.4 s for heuristic scheduling. Meanwhile, when the system size is increased from 10 to 20, the computing cost of optimal task scheduling increase from 3.16 to 6558.75 s while the computing cost of heuristic task scheduling increases from 0.07 to 0.25 s. From Fig. 13, we can also observe that both optimal scheduling and heuristic scheduling outperform random scheduling in each case.

7.3. Comparison of AppLeS and GHS on task scheduling

AppLeS is a well-known application-level task scheduling system for Grid environments [9–11]. Its scheduling decision is made based on the estimate of the application completion time. AppLeS predicts the application computation time with the formula $T = T_{\text{dedicated}}/AvailCPU$ where $T_{\text{dedicated}}$ denotes the application computation time in a dedicated resource and *AvailCPU* denotes is the prediction of the percentage of available CPU for this resource. The prediction of *AvailCPU* is provided by the Network Weather Service (NWS) [47]. Scheduling decision based on NWS prediction has two limitations. First, NWS targets short-term system-level performance prediction. As its claims, it is only suitable for jobs of five minutes time span or less. It cannot provide a satisfactory solution to long-term task scheduling. Secondly, NWS only provides resource availability, The effects of other system specific factors on the task execution time are not analyzed and thus not considered in task scheduling in AppLeS. In contrast, GHS makes scheduling decisions based on long-term application-level performance prediction. The effects of machine utilization, computing power, local job service, and task allocation on the completion time of a parallel task are individually identified.

An experiment is conducted to prove that short-term prediction cannot provide a satisfactory solution to long-term task scheduling. We first compare the prediction error of a long-term sequential application's completion time based on resource availability provided by NWS and GHS. Using the AppLeS formula $T = T_{\text{dedicated}}/AvailCPU$, we estimate different application completion times based on resource predictions provided by NWS in terms of 10 s (default set of NWS) and 5 min and provided by GHS, respectively. In GHS, since we assume that a remote task is assigned with a lower priority than local jobs, we can calculate *AvailCPU* with $AvailCPU = 1-\rho$ where the prediction of ρ is provided by the System-level Predictor. To have a thorough comparison, in this experiment, we calculate two performance metrics: percentage prediction error and square prediction error. Fig. 14 shows

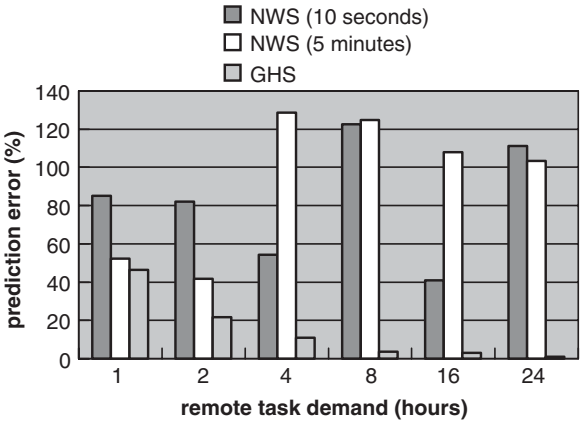

Fig. 14. Mean of the percentage prediction error based on NWS and GHS.

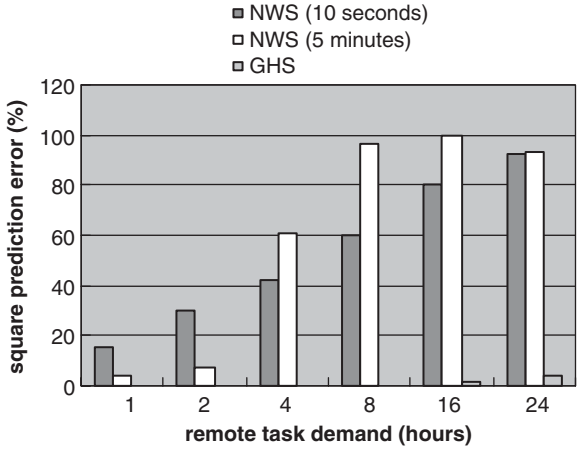

Fig. 15. Mean of the square prediction error based on NWS and GHS.

that the percentage prediction error based on NWS remains very high while the percentage prediction error based on GHS decreases with the increase in application workload. Fig. 15 shows that the square prediction error based on NWS is always higher than the square prediction error based on GHS. In this figure, the square prediction error is normalized with the function $\log_{1000}(1+(X-D_{\min})/(D_{\max}-D_{\min})^{*}999)$ where $D_{\max}$ is the maximum original square prediction error and $D_{\min}$ is the minimum original square prediction error. The results indicate that the estimation of the completion time of a large application based on short-term system prediction provided by NWS is far from satisfactory. In a short time range, the resource availability tends to be consistent with it history. So we can estimate a short-term application execution time based on the latest availability information. However, in a long time period, a large-scale application execution will be affected by arrivals and departures of local jobs. It is inappropriate to predict a long-term application performance using the short-term availability prediction.

Due to the short range of NWS prediction, AppLeS does not give a fixed scheduling for a large-scale parallel application. Instead, AppLeS adopts a multiple-phase scheduling approach for large scale meta-tasks. In other words, it constantly reschedules the tasks within the NWS prediction scope. At each scheduling event, NWS online prediction is used for task scheduling. This multiple-phase scheduling, in addition to the increased cost, has an inherited drawback. The later phases of online prediction are tampered by the assigned remote subtasks. Rescheduling based on the tampered resource availability is inappropriate. This situation would not happen in GHS because its prediction is made before the task scheduling. We have conducted an experiment to confirm that the multi-phase approach is not a quick fix for short-term prediction. A simulation environment is built following the descriptions of the application model and the Grid model in AppLeS [13]. The application has a set of independent subtasks. The input of each subtask is a set of files and a single file might be input to more than one subtask. The computational Grid is composed of a set of clusters of computing resources that are accessible to users via distinct network links. The estimate of file transfer time on each link is assumed to be available. In our experiment, the phase scheduling length is set at 500 s as used in AppLeS. The number of clusters and the map of input files onto subtasks are randomly generated in each simulation time. The system consists of 20 machines and the simulation runs 20 times for a given number of subtasks. We compare the average, minimum, and maximum of the task completion time (seconds) with two types of scheduling strategies. One is using the multiple-phase scheduling with NWS prediction. The other one is performing one-phase scheduling with GHS prediction. In both cases, we use a standard AppLeS scheduling algorithm, min–min heuristic [13]. The simulation results are summarized in Table 1. It shows that with GHS prediction the average task completion time decreases by 17–30% compared with that of NWS prediction. This is because, in multiple-phase scheduling, NWS online prediction is distorted by the execution of the meta-task at runtime.

The improvement of GHS is not due solely to the prediction. It is also due to its advanced scheduling algorithms. To compare the scheduling algorithms only, we modify AppLeS to let it access GHS' prediction. The comparison of task completion time (seconds) and the number of machine used with the two different scheduling systems is given in Table 2. The experimental results show that with GHS the task completion time decreases by 10–20% compared with that of AppLeS system while GHS only uses about one-half of the machines used in AppLeS. When the system size is 400, GHS uses 113 computers and achieves a better performance than AppLeS while AppLeS uses all machines. This is the limitation of the determined prediction approach used by AppLeS. It cannot identify the effect of the variation of resource availability on the application execution time. This indicates that GHS works better in a large distributed system. It has a real potential for Grid computing.

7.4. The efficiency of parameter measurement and GHS run-time cost

In GHS, we calculate the number of measurements for the next hour according to the system history over the previous 24 h.

Table 1
Comparison of multiple-phase scheduling with NWS prediction and one-phase scheduling with GHS prediction

Number of subtasks		250	500	1000	2000
GHS (s)	Average time	3892.0	6636.7	12819.4	24717.2
	Min. time	2869.9	5003.3	10743.7	21000.1
	Max. time	4671.2	7579.5	14516.4	29537.8
AppLeS (s)	Average time	4567.6	8553.2	16399.2	32121.2
	Min. time	3733.2	7298.3	14321.7	28180.9
	Max. time	5225.5	9557.9	18561.2	36627.1

Table 2
Comparison of task scheduling of AppLeS and GHS

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 used	13	26	57	99	113
AppLeS	Task completion time (s)	547.4	637.4	818.3	1022.7	1266
	Number of machine used	25	50	100	200	400

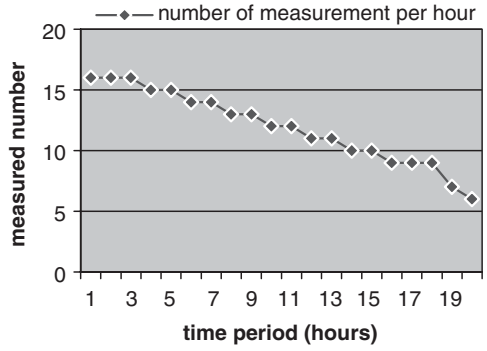

Fig. 16. Decreasing of measurement when the system is steady.

Table 3
The calculation time of the prediction component with different numbers of workstations

Nodes Number	4	8	16	32	64	128	256	512	1024
Time (s)	0.00	0.00	0.01	0.02	0.04	0.08	0.16	0.31	0.66

Our measurement mechanism can dynamically adjust the number of measurements to reduce the measurement cost. Fig. 16 shows an example of the fluctuation of the number of intervals measured in each hour when the machines are becoming steady. It indicates that the number of measurements decreases when the machine utilization remains at a certain level.

We have also measured the execution time of the prediction program. Table 3 gives the runtime of our prediction program on a Sun workstation, Scala. The results show that the run-time cost of our prediction component is 0.66 s when the number of machines is 1024. Compared to the potential gain from task scheduling, this run-time cost is negligible.

8. Conclusion and future work

We have presented the design and development of the GHS performance prediction and task scheduling system. GHS is uniquely designed for solving large-scale applications in a shared distributed environment. It consists of the components of performance monitor, system-level and application-level prediction, task partition and scheduling, and user interface. The underlying prediction models and scheduling algorithms have been discussed and experimental results have been given. Experimental and analytical results show that GHS adequately captures the dynamic nature of Grid computing. For large jobs, eight hours runtime or more, its prediction error is less than 10% on a single machine. The prediction error is even less on multi-processing systems. GHS optimal task scheduling provides a significantly better performance than that of existing methods, while its heuristic scheduling provides a near optimal performance. The contribution of GHS is two-fold, in both performance prediction and in task scheduling. We have shown that short-term prediction is not applicable to large scale applications, and have shown that the multi-phase prediction approach, which cuts a large application into many short-term time periods for performance prediction, is not a quick fix for large-scale task scheduling. The performance prediction and task scheduling system of GHS is significantly better than existing system for large-scale applications. For instance, compared with AppLeS, a widely used Grid scheduling system, GHS scheduling system decreases the task completion time by 17–30%, while using only about one-half of the machines used by AppLeS.

GHS is a long-term, application-level performance prediction and task scheduling system for non-dedicated distributed computing. It is a complement of existing Grid performance tools. It can be integrated into existing toolkits for better

service. For instance, NWS or RPS toolkits can be used to provide the performance measurement for GHS, or they can be combined with GHS to provide short-term system prediction. GHS can be combined with AppLeS to provide specific application scheduling. Like most existing performance systems, the current implementation of GHS has its limitations. For instance, GHS only considers the workload in distributed systems but not the communication and synchronization costs. NWS or the network bandwidth predictor [21] can be used to predict short-term bandwidth. Long-term network bandwidth modeling and prediction, however, is still a challenging task facing the network and distributed computing community as a whole. The Alpha version of GHS is available on-line at http://www.cs.iit.edu/∼scs/software.htm. The current implementation only demonstrates the feasibility and potential of the GHS approach. More work is needed to integrate GHS seamlessly into the Grid environment and with other existing Grid tools.

Acknowledgments

We would like to thank Dr. Gregor von Laszewski at Argonne National Laboratory and Dr. Kasidit Chanchio at Oak Ridge National Laboratory for their help in collecting the performance data. Dr. Von Laszewski is supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Project research and development. Globus Toolkit and Globus Project are trademarks held by the University of Chicago.

This research was supported in part by national science foundation under NSF grant EIA-0224377, CNS-0406328, ANI-0123930, and EIA-0130673.

References

[1] D. Abramson, R. Sosic, J. Giddy, B. Hall, Nimrod: a tool for performing parametised simulations using distributed workstations, in: Proceedings of the 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995, pp. 112–121.

[2] A. Acharya, G. Edjlali, J. Saltz, The utility of exploiting idle workstations for parallel computation, in: Proceedings of the 1997 ACM SIGMETRICS, Seattle, Washington, 1997, pp. 225–236.

[3] G. Allen, W. Benger, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, J. Shalf, The Cactus Code: a problem solving environment for the Grid, in: Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, 2000.

[4] G. Allen, W. Benger, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, J. Shalf, Cactus tools for Grid applications, Cluster Comput. 4 (2001) 179–188.

[5] Y. Amir, B. Awerbuch, A. Barak, R.S. Borgstrom, A. Keren, An opportunity cost approach for job assignment in a scalable computing cluster, IEEE Trans. Parallel Distrib. Systems 11 (2000) 760–768.

[6] R.H. Arpaci, A.C. Dusseau, A.M. Vahdat, et al., The interaction of parallel and sequential workloads on a network of machines, in: Proceedings of ACM SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, Ottawa, 1995, pp. 267–278.

[7] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, M. Yarrow, The NAS Parallel Benchmarks 2.0, Technical Report 95-020, NASA Ames Research Center, December 1995.

[8] A. Barak, S. Guday, R. Wheeler, The MOSIX Distributed Operating System, Load Balancing for UNIX, Lecture Notes in Computer Science, vol. 672, Springer, Berlin, 1993.

[9] F. Berman, A. Chien, K. Cooper, J. Dongarra, et al., The GrADS project: software support for high-level Grid application development, Internat. J. High Performance Comput. Appl. 15 (2001) 327–344.

[10] F. Berman, R. Wolski, H. Casanova, W. Cirne, et al., Adaptive computing on the Grid using AppLeS, IEEE Trans. Parallel Distrib. Systems 14 (2003) 369–382.

[11] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao, Application-level scheduling on distributed heterogeneous networks, in: Proceedings of Supercomputing' 96, Pittsburgh, PA, November 1996.

[12] H. Casanova, J. Dongarra, NetSolve: a network server for solving computational science problems, Internat. J. Supercomputer Appl. High Performance Comput. 11 (1997) 212–223.

[13] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman, Heuristics for scheduling parameter sweep applications in Grid environments, in: Proceedings of the Ninth Heterogeneous Computing Workshop, 2000, pp. 349–363.

[14] H. Casanova, G. Obertelli, F. Berman, Rich Wolski, The AppLeS parameter sweep template: user-level middleware for the Grid, in: Proceedings of Super Computer 2000, Dallas, TX, November 2000.

[15] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, Gaurang Mehta, et al., Mapping abstract complex workflows onto Grid environments, J. Grid Comput. 1 (2003) 25–39.

[16] P. Dinda, 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.

[17] P. Dinda, D. O'Hallaron, Host load prediction using linear models, Cluster Comput. 3 (2000) 265–280.

[18] Distributed Optical Testbed (DOT), ⟨http://www.dotresearch.org/⟩.

[19] A.B. Downey, A parallel workload model and its implications for processor allocation, J. Cluster Comput. 1 (1998) 133–145.

[20] C. Du, S. Ghosh, S. Shankar, Xian-He Sun, A runtime system for automatic rescheduling of MPI Program, in: 33rd International Conference on Parallel Processing, Montreal, Que., Canada, August 15–18, 2004.

[21] A. Eswaradass, X.-H. Sun, M. Wu, A neural network based predictive mechanism for available bandwidth, in: Proceedings of the 19th International Parallel and Distributed Processing Symposium, Denver, CO, April 2005.

[22] I. Foster, C. Kesselman, The Grid2: Blueprint for a New Computing Infrastructure, Morgan-Kaufman, San Francisco, CA, 2004.

[23] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Francisco, CA, 1999.

[24] I. Foster, A. Iamnitchi, On death, taxes, and the convergence of peer-to-peer and Grid computing, in: Proceedings of the Second International Workshop on Peer-to-Peer Systems, Berkeley, CA, 2003.

[25] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[26] I. Foster, A. Roy, V. Sander, A quality of service architecture that combines resource reservation and application adaptation, in: Proceedings of the Eighth International Workshop on Quality of Service, Pittsburgh, PA, 2000, pp. 181–188.

[27] J. Frey, T. Tannenbaum, I. Foster, S. Tuecke, Condor-G: a computation management agent for multi-institutional Grids, in: Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing, San Francisco, CA, August 2001.

[28] Globus Toolkit, ⟨http://www.globus.org/toolkit/⟩.

[29] L. Gong, X.-H. Sun, E.F. Waston, Performance modeling and prediction of non-dedicated network computing, IEEE Trans. Comput. 51 (2002) 1041–1055.

[30] M. Harchol-Balter, A. Downey, Exploiting process lifetime distributions for dynamic load balancing, ACM Trans. Comput. Systems 15 (1997) 253–285.
[31] R. Jain, The Art of Computer Systems Performance Analysis, Wiley, New York, 1992.
[32] J.A. Kaplan, M.K. Nelson, A comparison of queuing, cluster, and distributed computing systems, NASA TM 109025, June 1994.
[33] L. Kleinrock, W. Korfhage, Collecting unused processing capacity: an analysis of transient distributed systems, IEEE Trans. Parallel Distrib. Systems 4 (1993) 535–546.
[34] T. Kosar, M. Livny, Stork: making data placement a first class citizen in the Grid, in: Proceedings of the 24th IEEE International Conference on Distributed Computing Systems, Tokyo, Japan, March 2004.
[35] S.T. Leutenegger, X.-H. Sun, Limitations of cycle stealing of parallel processing on a network of homogeneous machines, J. Parallel Distrib. Comput. 43 (1997) 169–178.
[36] M.J. Lewis, A. Grimshaw, The core Legion object model, in: Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society Press, Los Alamitos, California, August 1996.
[37] M. Litzkow, M. Livny, M. Mutka, Condor—a hunter of idle workstations, in: Proceedings of the Eighth International Conference of Distributed Computing Systems, San Jose, 1988, pp. 104–111.
[38] B.P. Miller, M.D. Callaghan, et al., The Paradyn parallel performance measurement tools, IEEE Computer 28 (1995) 37–46.
[39] M. Mutka, M. Livny, The available capacity of a privately owned machine environment, Performance Evaluation 12 (1991) 269–284.
[40] R. Raman, M. Livny, M. Solomon, Matchmaking: distributed resource management for high throughput computing, in: Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, IL, July 28–31, 1998.
[41] S. Shende, A.D. Malony, J. Cuny, K. Lindlan, P. Beckman, S. Karmesin, Portable profiling and tracing for parallel scientific applications using C++, in: Proceedings of SPDT'98: ACM SIGMETRICS Symposium on Parallel and Distributed Tools, Welches, Oregon, August 1998, pp. 134–145.
[42] X.-H. Sun, M. Wu, Grid Harvest Service: A system for long-term, application-level task scheduling, in: Proceedings of 2003 IEEE International Parallel and Distributed Processing Symposium, Nice, France, April 2003.
[43] V. Taylor, X. Wu, J. Geisler, R. Stevens, Using kernel couplings to predict parallel application performance, in: Proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing, Edinburgh, Scotland, 2002.
[44] H.L. Truong, T. Fahringer, G. Madsen, A.D. Malony, H. Moritsch, S. Shende, On using SCALEA for performance analysis of distributed and parallel programs, in: Proceedings of the ACM/IEEE SC2001 Conference, Denver, CO, USA, November 2001.
[45] R. Vilalta, C.V. Apte, J.L. Hellerstein, S. Ma, S.M. Weiss, Predictive algorithms in the management of computer system, IBM Systems J. 41 (2002) 461–474.
[46] R. Wolski, Dynamically forecasting network performance using the network weather service, Cluster Computing 1 (1998) 119–132.
[47] R. Wolski, N.T. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, J. Future Generation Comput. Systems 15 (1999) 757–768.
[48] M. Wu, X.-H. Sun, A general self-adaptive task scheduling system for non-dedicated heterogeneous computing, in: Proceedings of IEEE Cluster Computing Conference, Hong Kong, 2003, pp. 354–361.

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. Currently, he is a professor and the director of the Scalable Computing Software Laboratory in the Computer Science Department at Illinois Institute of Technology. Dr. Sun's research interests include Grid and cluster computing, software system, performance evaluation, and pervasive and scientific computing. His research has been supported by DoD, DoE, NASA, NSF, and other government agencies. He received the ONR and ASEE Certificate of Recognition award in 1999, the Best Paper Award from the International Conference on Parallel Processing (ICPP01) in 2001, and the Best Poster Award from the IEEE Supercomputing Conference in 2003.