The GHS Grid Scheduling System: Implementation and Performance Comparison

Ming Wu, Xian-He Sun

Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60616, USA
{wuming, sun}@iit.edu

Abstract

Effective task scheduling and deployment is hard to achieve in a Grid environment, where computing resources are heterogamous and shared between local and Grid users without a central control. Current scheduling systems, such as AppLeS, use NWS (Network Weather Service) for short-term estimation of resource availability and do not address the influence of the variation of resource availability in task scheduling. These inherent limitations prevent existing scheduling systems from working effectively to solve large-scale tasks in a Grid environment. Adopting APST (AppLeS Parameter Sweep Template) as the deployment environment, we have developed a task scheduling system for large-scale applications based on our recent results in performance prediction and task scheduling. Preliminary experimental results show that the newly developed system works well and is significantly more appropriate for large applications than existing systems.

1. Introduction

Evaluating the effect of resource sharing on application performance is a major challenge of task scheduling in shared environments. In a shared environment, a remote task competes with local jobs, or even other remote tasks, for resources. The completion time of a task is not only determined by its workload but also the availability of resources. Scheduling systems, such as AppLeS, PEGASUS, CONDOR-G, CHIMERA, and Stork, have been developed recently for distributed shared environments. In these systems, scheduling decision is made based on the prediction of application performance with the consideration of the CPU availability, which is estimated by NWS (Network Weather Service) [1]. However, NWS is designed for short-term predictions (up to 5 minutes as it claims) and cannot provide appropriate predictions for long-term applications. AppLeS [2, 3] is one of the most used Grid scheduling systems and has made extra efforts to adopt NWS for large scale applications. To compensate the known flaw of NWS prediction, AppLeS conducts periodic rescheduling. That is, an application is decomposed into or predefined as a set of numerous small subtasks and then the scheduling is re-performed periodically in small time period based on short-term predictions. Rescheduling involves task redeployment, while it mitigates the cost of inaccurate predictions, it adds in data movement cost. In addition, AppLeS scheduling decision is made based on minimizing each individual task completion time, where minimizing each task does not necessarily lead to an optimized completion time of a parallel application. It does not consider the variation of resource availability during each task deployment, other than repeating prediction/scheduling. To address these problems, we have introduced a new performance modeling technique to evaluate the effect of resource contention on the application performance and developed new scheduling algorithms [4]. A performance prediction system, named Grid Harvest Service (GHS), is thus proposed. Instead of using only utilization to describe the resource availability, GHS models the resource usage pattern with an M/G/1 queue system. The effects of machine utilization, computing power, local job service, and task allocation on the application performance are individually identified.

Even after a scheduling decision is made, deploying a distributed task in a Grid computing environment still requires an integrated system solution. The deployment involves job submission, data movement, and resource monitoring. It is not an easy task in a shared environment, where the computing can cross multiple administration domains, which adopt different resource management middleware and policies. APST, a production application execution environment, is developed based on a component-oriented design and thus can be integrated with different middleware or components [2]. Adopting an enhanced APST as the deployment environment, we

1-4244-0054-6/06/$20.00

have developed a task scheduling system based on our recent results in performance prediction and task scheduling [4]. Preliminary experimental results show that the new system works well and is significantly more appropriate for large applications than existing systems.

2. Related Work

Job management has been studied intensively to develop efficient schedule and monitor systems for parallel and distributed computing. Existing systems, including LSF, PBS Pro, Sun Grid Engine/CODINE, and Maui Scheduler, are mainly designed for improving the utilization of expensive resources in high-performance computing and assume a relatively dedicated and stable computing environment. Issues such as load balance and fairness are often considered in the design and development of these systems instead of an individual job's execution time.

Research on scheduling jobs in shared environments has attracted extensive attention recently. Condor system [5] provides a matchmaking mechanism to allocate resources with ClassAds. The scheduling strategy is based on the match of the users' specification of their job requirements and preferences, with the machines' characteristics, available time periods, and conditions. AppLeS uses a loop of task events to schedule subtasks of a meta-task dynamically (a meta-task is composed of a group of independent and indivisible subtasks) [2, 3]. AppLeS supports Min-min, Max-min, Sufferage, and Xsufferage heuristic algorithms with the consideration of file transfer cost. The APST software was developed to make easy to deploy and adaptively schedule meta-tasks in distributed environments. With APST, AppLeS can be easily deployed in Grid environments and gain popularity. However, the scheduling of AppLeS has three limitations. First, its estimation of task execution time is based on the prediction provided by NWS. NWS is designed for short-term predictions (up to 5 minutes as it claims) and cannot provide appropriate prediction for long-term applications. As a result, AppLeS conducts periodic scheduling to compensate the flaw of prediction. This rescheduling method assumes an application can be partitioned into or composed of a group of fine-grained subtasks to fit into the short-term resource availability prediction (the default scheduling period is 500 second in APST). Second, the AppLeS scheduling plan is generated based on minimizing each individual task's completion time where minimizing each task does not necessarily lead to an optimized completion time of a parallel application. Third, AppLeS makes the scheduling decision based on the determined resource availability estimation. It doesn't consider the impact of the variance of resource availability on application performance. Recently, Yang and Casanova have proposed a multi-round algorithm for scheduling parallel application with divisible workloads [6]. A software, named APST-DV has been developed based on the multi-round scheduling [7]. However, their new work is based on the assumption that the computation capacit of machines and the communication speed of network links are fixed in each round. This methodology is more appropriate for a dedicated system instead of a shared environment. In PEGASUS and CHIMERA workflow management systems [8, 9], data duplication is considered in mapping an abstract workflow onto Grid environments. Their recent study in [10] also indicates that minimizing the completion time of each individual part of a workflow does not necessarily lead to the optimization of the whole workflow's completion time.

3. GHS-APST Software

Short-term prediction is hardly appropriate for long-term scheduling. To address these problems, we have introduced new performance modeling techniques for evaluating the effect of resource contention on a long-term application's performance. A series of task scheduling algorithms and resource measurement and performance data collection methodologies have been designed and developed accordingly. The Grid Harvest Service (GHS) system has been proposed to enable the collaboration among performance measurement, performance modeling, and task scheduling. The Alpha version of GHS is implemented and available on-line at http://www.cs.iit.edu/~scs/software.htm. An early report on the GHS performance prediction system can be found at [4]. This study extends the previous results to fully develop the GHS software system for task scheduling.

3.1. GHS Software

The Grid Harvest Service is a performance and task scheduling system for Grid computing. Its major subsystems include performance evaluation, task allocation, and task scheduling. Coordinately, these subsystems provide appropriate services to harvest Grid computing.

The major goal of the performance evaluation subsystem is to estimate resource availability and to find its influence on application performance. Thus this subsystem consists of two parts: Application-level Predictor and System-level Predictor. The goal of Application-level Predictor is to estimate the application completion time, with a given application workload distribution over a set of resources. The System-level Predictor estimates the resource availability. In a shared machine, application performance is mainly decided by the available computation power of the shared resource.

To identify the impact of resource availability on application completion time, we model the resource usage pattern with an M/G/1 queue system. It considers the heterogeneous machine utilization and computing capacity, heterogeneous job arrival rate as well as heterogeneous service distribution. The model was derived from a combination of rigorous mathematical analysis and intensive simulation to make it generic and practically useful. For more detailed information, people can refer to our previous work in [11, 4].

The major motivation of distributed computing is that it can provide the "integrated" computation power that is needed for solving large-scale scientific applications by connecting numberous resources. The required computing power can not be satisfied by any single computer. Correspondently, the first thing of running these applications in distributed computing is to decide how to partition an application into subtasks and then maps them to a chosen set of resources for optimal performance via task scheduling. In GHS, this task is performaned by the task allocation subsystem. Considering the heterogeneous and dynamic resource availability and capacity in shared distributed environments such as Grid computing, we develop a mean-time task partition algorithm to distribute the workload of a parallel program to each resource so that the difference of the mean of expected execution times of the subtasks is minimal instead of using the conventional workload balance approach widely applied in parallel computing. The task allocation subsystem of GHS partitions the workload of parallel applications with respects of CPU, memory, network resource heterogeneity and resource sharing [12].

The task scheduling subsystem of GHS 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 implements different scheduling algorithms for Grid computing according to the application's requirement, such as a single task scheduling, parallel processing, and meta-task scheduling. Based on the prediction from the performance evaluation subsystem and task partition from the allocation subsystem, it checks potential available resources in the system and then searches for the best set of resources to assign the application. Two searching methodologies are supported: heuristic and optimal. A heuristic search is to find a near optimal solution with a reasonable cost. A task-rescheduling algorithm is also implemented to handle different abnormal situations, such as abnormal computing and I/O performance, system shut down, un-trusted system behavior, failure of Globus software package, and undermined security. The subtasks of an application on resources showing abnormal performance are assigned to other appropriate resources based on a re-estimation of the application completion time. The task reallocation is enabled by the High Performance Computing Mobility (HPCM) [13].

In the implementation of the Alpha version of GHS, we fully decouple these subsystems. They are presented to users as normal UNIX utilities. Each subsystem can be independently integrated into other performance evaluation and task scheduling tools.

3.2. APST Execution Management

Resources in Grid computing are usually heterogeneous and geographically distributed and are administrated by different domains. Much effort is being made to establish standard, open, general-purpose protocols and interfaces for resource sharing and coordination [14]. Many middleware are developed for different concerns in a general network computing: security, information sharing, file transfer, and remote execution. The interaction with these middleware is essential for task deployment in Grid environments.

APST stands for AppLeS Parameter Sweep Template [15]. It is a production application execution environment. It was developed in University of California, San Diego and originally designed for scheduling distributed implementations of "Parameter Sweep Applications" in Grid environments. The parameter sweep applications are structured as a set of "experiments", each of which is executed independently with different set of parameters. The strengths of APST are two-fold: flexible software architecture and easy-of-use. APST software is decoupled into four components: Controller, Scheduler, Actuator, and Metadata Bookkeeper. Standard APIs are provided for the interaction among these components while their implementations can be different to adapt to the underlying Grid resources and middleware. APST can be easily deployed in a wide range of distributed environments and integrated with a number of middleware. For example, it can utilize various data transport mechanisms (GASS, IBP, GridFtp, scp, or NFS) to transfer input and output files among different sites and different task execution mechanisms (ssh, NetSolve, GRAM, Condor) to run applications on remote resources.

APST is easy-to-use. The software is composed of a client, apst, and a daemon, apstd. The apstd handles task assignment, job execution, and data file transfer. The apst is a client program, an interface portion that allows users to communicate with apstd. Through apst, a user can add new resources/tasks, check current status of resources/tasks, and stop the execution of apstd. APST uses a XML file to describe tasks and resources. Each task description specifies the task's executable and its required command-line arguments, the relative computational cost, its input and output files. Each resource (compute, storage) description specifies the location and the way to access. The XML file also

includes a gridinfo element, which is used to specify the source of information about the resources.

4. Integration GHS Prediction and Scheduling with APST

APST has been developed over years and tested with many applications, such as Mcell, Volume Rendering, Encyclopedia of life, in Grid computing. A common characteristic of these applications is that they can be partitioned into or composed of fine-grained subtasks so that AppLeS scheduling supported in APST can estimate their subtask completion time with the short-term prediction of resource availability provided by NWS. This limits the application of APST system and affects the scheduling algorithm design. To remove this limitation, we integrate GHS prediction and scheduling with APST. APST daemon consists of four components: Metadata Bookkeeper, Scheduler, Actuator, and Controller. In the following, we introduce their major functionalities and then present the modification component by component.

Meta-data Bookkeeper is in charge of accessing resource meta-data. The Bookkeeper component has a facility called Elagi. It extracts performance metrics of resources from a variety of information services, such as NWS, MDS, or Ganglia. The currently supported performance metrics are CPU count, CPU load, network bandwidth and latency. In GHS, we model the resource usage pattern with an M/G/1 queue system to evaluate the impact of resource availability on the performance of a long-term application. The job arrival rate, job service time standard deviation, and resource utilization, have to be collected in order to calculate the mean, the standard deviation, and the distribution of applications completion time in a shared environment. Among them, only resource utilization is supported as a performance metric in APST in the term of ELMETRIC_CPU_LOAD. To integrate GHS prediction into APST, we first modify the MetricType and ServiceType data structure in the Meta-data Bookkeeper. Job arrival rate and job service time standard deviation are included to reflect the new performance modeling. To support the retrieve of these two new resource metrics, we add a GHS server information service. It interacts with GHS Performance Measuring Engine (PME) and System-level Predictor (SLP) on each machine to collect resource information. The updated MetricType and ServiceType data structure is given in Figure 1. The added ServiceType, ELMETRIC_GHS, indicates a new information service is supported. Correspondingly, two functions, GhsFillKey and GhsFillMetric, are added in Elagi.

Scheduler implements several APST scheduling algorithms, maxmin, minmax, sufferage and Xsufferage,

```
/* Supported Grid metric services */
typedef enum {
  ELMETRIC_LOCAL, ELMETRIC_GANGLIA,
    ELMETRIC_MDS, ELMETRIC_NWS,
    ELMETRIC_GHS,
    ELMETRIC_METRIC_SERVICE_COUNT
} ELMETRIC_MetricServiceTypes;

/* Supported Grid metrics */
typedef enum {
 ELMETRIC_CPU_COUNT,
  ELMETRIC_CPU_LOAD,
 ELMETRIC_TCP_BANDWIDTH,
 ELMETRIC_TCP_LATENCY,
 ELMETRIC_JOB_ARRIVAL_RATE,
 ELMETRIC_SERVICE_TIME_DEVIATION,
 ELMETRIC_METRIC_COUNT,
} ELMETRIC_MetricTypes;
```

Figure 1. Extended MetricTypes and MetricServiceTypes

besides the workqueue algorithms. These algorithms generate a scheduling plan based on the user's submitted task and resource information. The scheduling plan is then executed by Actuator. In the Scheduler component, APST uses a procedure, FillGanttCharts(), to generate the scheduling plan. We enhance this procedure by adding a new API for meta-task scheduling, named GhsMetataskSched(). In GhsMetataskSched(), the Task Allocator is first called to partition a meta-task into a cluster of subtask groups and then the Application-level Predictor is called for the estimation of the application completion time for a given task partition plan. This process is repeated until an optimal or near-optimal scheduling plan is found. The generated scheduling plan is still stored in the GanttChart data structure and thus the execution of Actuator is not affected.

Actuator implements two standard APIs, env_api and transport_api to handle task execution and file transfer on Grid resources accessible to different middleware software. It executes the scheduling plan generated by Scheduler. We keep this component intact in our modification.

Controller interacts with the APST client to accept and response user's request. One of the major functionalities of Controller is to parse the XML file to collect the task and resource information and store them into correspondent hash tables where the other components can access. Because a new information service, GHS server, is supported, we modify the ProcessXmlFile() function in the Controller component so that the new information service can be parsed in the XML file.

Figure 2 gives the software architecture of the updated APST. In this Figure, components from GHS are marked with the orange color; components newly added or modified to facilitate the integration are marked with the green and yellow color respectively; components from APST are marked with the grey color.

5. Experimental Results

We have conducted experiments to compare GHS scheduling and AppLeS scheduling. An experiment is first carried to estimate the execution time of a long-term application based on resource availability prediction provided by NWS and GHS. This experiment is used to verify that short-term prediction cannot provide a satisfactory solution to long-term task scheduling. We then test the efficiency of GHS long-term prediction with a real Grid application, Cactus, on an actual Grid environment. After that, we compare application execution time using the AppLeS's multi-phase scheduling approach with NWS prediction and the one-phase scheduling approach with GHS prediction. The better performance of GHS is not only due to its long-term performance prediction but also due to its advanced scheduling algorithms. Finally, an experiment is conducted to compare the AppLeS and GHS scheduling algorithms only.

The test platforms used in our experiments are the *Sunwulf* cluster and the DOT Grid Testbed [16]. *Sunwulf* is a heterogeneous 84-node Sun ComputeFarm at IIT. The DOT connects clusters at ANL, NCSA, NU, UC, UIC, and IIT via the advanced "I-Wire" network. Each cluster is composed of one sever and multiple computing nodes. Local jobs on each resource in these test platforms are simulated with different job arrival rates and service rates, which follows the observation of over one million real-life processes generated from different academic workloads in Berkeley, as well as the machine usage pattern observation by researchers at Wisconsin-Madison, Maryland, Carnegie Mellon, et al [17]. Cactus [18], a numerical simulation of a 3D scalar field produced by two orbiting astrophysical sources, is used as our test application. In the test, we use the parallel version of Cactus application. It decomposes the 3D scalar field into sub-fields and each sub-field along with a small overlap region is mapped onto a different processor in the cluster

5.1. Long-term Task Scheduling Based on Short-term Resource Prediction

AppLeS' scheduling decision is made based on the estimate of the application completion time [2, 3]. AppLeS predicts the application computation time with the formula $T = T_{dedicated} / AvailCPU$ where $T_{dedicated}$ denotes the application computation time in a dedicated resource and $AvailCPU$ denotes the prediction of the percentage of available CPU for this resource. The prediction of is provided by the NWS [1]. 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.

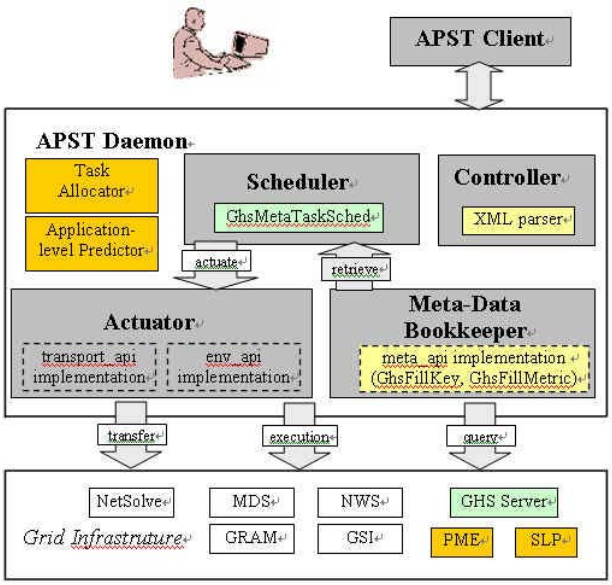

Figure 2. Software architecture of updated APST

An experiment is conducted to confirm that short-term prediction cannot provide a satisfactory solution to long-term task scheduling. We compare the prediction error of the application completion time on the *Sunwulf* ComputeFarm based on resource availability provided by NWS and GHS. The applications are NAS Benchmarks (BT, CG, LU, MG, IS and SP, with the class type of "A" or "W"). Using the AppLeS formula $T = T_{dedicated} / AvailCPU$, we estimate different application completion times based on resource predictions provided by NWS in terms of 10 seconds (default set of NWS) and 5 minutes 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 System-level Predictor.

To have a thorough comparison, in this experiment, we calculate two performance metrics: percentage prediction error and square prediction error. Percentage prediction error is defined as

$|(\text{Prediction} - \text{Measurement})/\text{Measurement}|$ [1]. Square prediction error is defined as $(\text{Prediction} - \text{Measurement})^2$ [1, 19]. Both performance metrics are generally used in the literature to evaluate the prediction accuracy. Figure 3 shows 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. Figure 4 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 prediction of resource availability provided by NWS is far from satisfactory no matter which metric is used.

To verify the efficiency of GHS application-level prediction in actual Grid environments, we have tested the Cactus application on the DOT Grid Testbed. 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. Two factors influence the accuracy of this kind actual testing, workload determination and system software interference. As a user we only can estimate the workload of Cactus based on its iteration number and input values – which is error prone. Also the underlying DOT management system may take CPUs away from time to time. Nevertheless, Figure 5 shows the model working well even with these two factors.

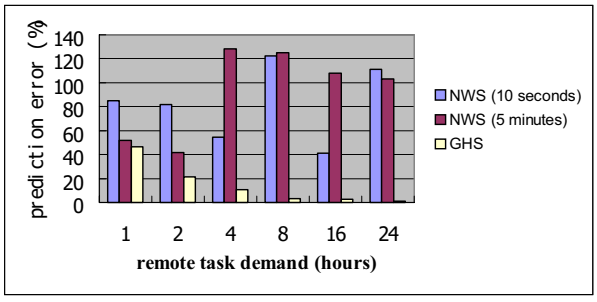

Figure 3. Mean of the percentage prediction error based on NWS and GHS

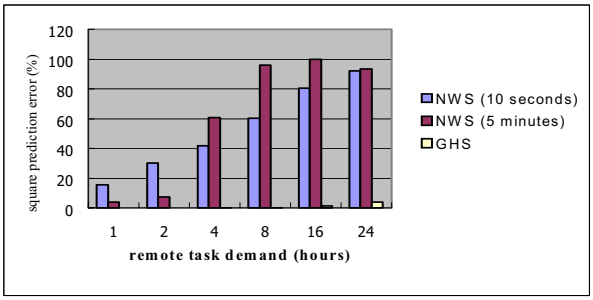

Figure 4. Mean of the square prediction error based on NWS and GHS

5.2. Multiple-phase Scheduling with NWS Prediction and One-phase Scheduling with GHS Prediction

Due to the short range of NWS prediction, AppLeS doesn't 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 wouldn't 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 Grid environment simulator was built to compare the performance of GHS scheduling with AppLeS scheduling. The reason why we use a simulator is because of the difficulty to access a large-scale, appropriate Grid environment to conduct the testing. The simulated Grid environment is composed of a number of clusters of machines, which can be scaled from dozens of resources to thousands of resources. We set different job arrival rates and service rates for these machines to simulate heterogeneous machine usage patterns in a Grid environment. The local job lifetime is generated with the bounded Pareto distribution [Balt02]. 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. In our experiment, the phase scheduling length is set at 500 seconds as used in AppLeS. The number of clusters and the map of input files onto

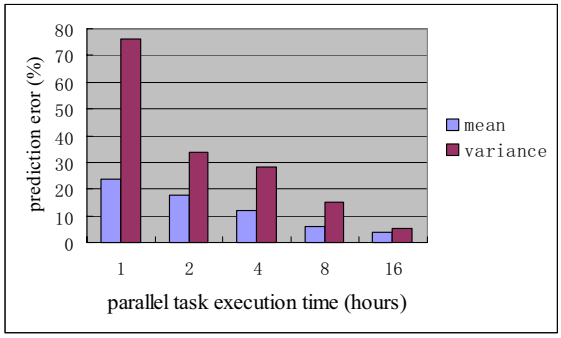

Figure 5. Mean and variance of prediction error of Cactus's execution time

subtasks are randomly generated in each simulation time. The system consists of 20 machines. 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. 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.

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

Table 2. Comparison of AppLeS and GHS scheduling (machine number and task completion time)

Workload (Max. machine number)		13801.7 (25)	27619.2 (50)	53779.5 (100)	108642.5 (200)	215141.0 (400)
GHS	task time (s)	496.4	557.7	712.8	874.5	1140.4
	number	13	26	57	99	113
AppLeS	task time (s)	547.4	637.4	818.3	1022.7	1266
	number	25	50	100	200	400

5.3. Comparison of AppLeS and GHS on Task Scheduling

We have shown that GHS scheduling outperforms AppLeS scheduling because short-term resource prediction cannot satisfactorily support the scheduling of long-term applications. What will happen if AppLeS is supported with a long-term resource prediction? To answer this question, we repeat the above simulation except that both scheduling algorithms are provided with the same resource 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 because AppLeS scheduling uses a determined approach to estimate the application execution time, $T = T_{dedicated} / AvailCPU$. This deterministic estimation of application performance may work well for single task scheduling. However, it has limitation for parallel task scheduling. In parallel processing, the application execution time is decided by the maximum subtask completion time. The effect of variation of resource availability on application performance should be considered. With more resources, the chance of inaccurate availability prediction at one machine (due to resource variation) will be higher, which leads to a longer application execution time than the estimated using the deterministic approach. In contrast, GHS uses M/G/1 to describe the machine usage pattern of each resource. The application performance is expressed with a cumulative density distribution (CDF) of the application execution time. The impact of resource availability variation on application performance is reflected in this CDF expression. Thus GHS provides a more accurate prediction of application performance in parallel processing. This indicates GHS scheduling algorithms can provide a better scheduling decision than AppLeS scheduling algorithms, especially in a large-scale system like a Grid.

6. Conclusions

Task scheduling requires an integrated solution of performance prediction, scheduling algorithms, and system development. Due to this reason, developing a suitable and broadly applicable scheduling system has been elusive. This is especially true for Grid computing, where computing resources are shared, local jobs are autonomic, and Grid tasks are distributed and deployed under different middleware and local management systems. Based on our previous results in performance prediction and task scheduling, in this study we have presented the development of the GHS (Grid Harvest Service) task scheduling system for Grid computing. We have introduced the system structure of GHS and the implementation of its major components; in particular we discussed the task deployment component. The performances of GHS are analyzed and compared with existing systems.

GHS consists of the GHS performance prediction, GHS task scheduling, and APST task execution environment. GHS performance prediction identifies the impact of resource sharing on the application execution time. GHS task scheduling selects the best machine set for execution. APST is a production application execution

environment. It can be integrated with a number of middleware for job submission, data movement, and resource monitoring on Grid environments. To integrate GHS prediction and scheduling with APST, we have modified the three major components of APST: Metadata Bookkeeper, Scheduler, and Controller. A meta-task scheduling algorithm based on the GHS performance modeling is added into the APST Scheduler to support long-term application scheduling. A GHS Server component is also developed and integrated into the APST Metadata Bookkeeper to provide the performance metric information retrieving needed for performance modeling. Experiments and simulations are conducted to verify the accuracy and feasibility of the new developed system. The experimental results show that the GHS scheduling system outperforms the AppLeS scheduling system in both the scheduled application performance and the number of occupied resources. The experiment with the Cactus application on the DOT testbed shows that the GHS prediction works well in an actual Grid environment. By introducing APST into GHS scheduling, both fine-grained meta-tasks and coarse-grained meta-task can be scheduled with the newly developed GHS task scheduling system. An alpha version of the GHS task scheduling system has been developed and released (see http://www.cs.iit.edu/~scs/software.htm). More experimental tests and refinements will be conducted in the future work.

Acknowledgments

This research was supported in part by national science foundation under NSF grant SCI-0504291, CNS-0406328, and EIA-0224377.

References

[1] R. Wolski, Dynamically forecasting network performance using the network weather service, *Cluster Computing*, 1 (1998) 119-132.

[2] H. Casanova, G. Obertelli, F. Berman, Rich Wolski, The AppLeS parameter sweep template: user-level middleware for the Grid, *in the Proc. of SuperComputing'2000*, Dallas, TX, November 2000.

[3] F. Berman, R. Wolski, H. Casanova, W. Cirne, et al, Adaptive computing on the Grid using AppLeS, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 4, pp 369-382, 2003.

[4] X.-H. Sun and M. Wu, Grid Harvest Service: a system for long-term, application-level task scheduling, *in the Proc. of 2003 IEEE International Parallel and Distributed Processing Symposium*, Nice, France, April, 2003.

[5] Rajesh Raman, Miron Livny, and Marvin Solomon, Matchmaking: Distributed Resource Management for High Throughput Computing, *in the Proc. of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28-31, 1998, Chicago, IL.

[6] Y. Yang, H. Casanova, UMR: A multi-round algorithm for scheduling divisible workloads, *in the Proc. of the International Parallel and Distributed processing Symposium*, Nice, France, April 2003.

[7] K. van der Raadt, Y. Yang, H. Casanova, Practical divisible load scheduling on Grid platforms with APST-DV, *in the Proc. of 2005 IEEE International Parallel and Distributed Processing Symposium Denver*, CO, April 2005.

[8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, Gaurang Mehta, et al. Mapping abstract complex workflows onto Grid environments, *Journal of Grid Computing*, 1 (2003) 25-39.

[9] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, Chimera: A virtual data system for representing, querying, and automating data derivation, *in the Proc. of the 14th Conference on Scientific and Statistical Database Management*, 2002.

[10] J. Blythe, S. Jain, E. Deelman, Y. Gil, et al, Task Scheduling Strategies for Workflow-based Applications in Grids, *in the Proc. of CCGrid 2005*, Cardiff, UK, 2005.

[11] L. Gong, X.H. Sun, and E. F. Waston, Performance modeling and prediction of non-dedicated network computing, *IEEE Trans. on Computers*, Vol. 51, No. 9, pp. 1041-1055, September, 2002.

[12] M. Wu, and X.-H. Sun, Memory conscious task partition and scheduling in Grid environments, *in the Proc. of 5th IEEE/ACM International Workshop on Grid Computing (in conjunction with SC 2004)*, Pittsburgh, Nov. 2004.

[13] C. Du, X.-H.Sun, and K. Chanchio, HPCM: a pre-compiler aided middleware for the mobility of legacy code, *in the Proc. of IEEE International Conf. on Cluster Computing*, 2003, Hong Kong, Dec. 2003.

[14] I. Foster and C. Kesselman, The Grid2: Blueprint for a New Computing Infrastructure, Morgan-Kaufman, 2004.

[15] AppLeS Parameter Sweep Template, http://grail.sdsc.edu/projects/apst/.

[16] Distributed Optical Testbed, http://www.dotresearch.org/

[17] M. Harchol-Balter, "Task Assignment with Unknown Duration," *J. ACM*, Vol. 49, No. 2, pp. 260-288, 2002.

[18] 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 Computing*, 4 (2001) pp. 179-188, 2001.

[19] P. Dinda, D. O'Hallaron, Host load prediction using linear models, *Cluster Computing*, 3 (2000) 265-280.