GHS: A Performance System of Grid Computing

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

Abstract

Conventional performance evaluation mechanisms focus on dedicated distributed systems. Grid computing infrastructure, on another hand, is a shared collaborative environment constructed on autonomic virtual organizations. The non-dedicated characteristic of Grid computing prevents the leverage of conventional task scheduling systems. In this study, we present the design and development of the Grid Harvest Service (GHS) performance evaluation and task scheduling system for solving large-scale applications in a shared network environment. GHS combines stochastic models and artificial intelligence learning mechanisms with task scheduling algorithms. It considers both computing and network contention and supports scheduling for single task, parallel processing, and meta-tasks. Experimental results show that GHS provides a satisfactory solution for performance prediction and task scheduling and has a real potential.

1. Introduction

With the advance of Internet, many scientists turned to construct large geographically distributed systems in recent years. The successes of distributed systems such as Condor, NetSolve, Globus, and Nimrod inspire and facilitate the formation of national scale distributed environments, Grid computing [FoKe04]. While much progress has been made in standardization of protocols and interface to facility coordination, the main challenge of enterprise network computing remains the same: resource management and task scheduling. There is a big gap between the potential peak performance and the delivered performance in Grid computing.

The conventional parallel processing scheduling methods cannot apply directly to a Grid environment where computing resources are autonomic shared. The key to Grid task scheduling is to understand the usage pattern and predict the availability of computing and communication resources, and to find their influence on the application performance. Some latest Grid tools, such as Network Weather Service (NWS) [WoSH99], have been developed to meet the need. However, these tools are for short-term resource availability (generally in tens of seconds). Another solution is adapting resource reservation to reduce the complexity of resource management in a shared environment. This approach requires resource owners to have good planning on their own tasks and suffers in system utilization. This approach is useful for high priority tasks or to show the potential of Grid computing, it but has difficulties to be employed in a general enterprise environment. Also, resources reservation will be more effective if it is based on resource availability prediction. Supported by the NSF NGS program, we have been developing a long-term, application-level performance prediction and task scheduling system, the Grid Harvest Service (GHS) system, for Grid computing. The "long-term" signifies that the system is designed for large applications that need hours of computations and is in contrast to the current Grid performance systems such as NWS. By "application-level" we mean that the goal is to reduce the run-time of user applications and is in contrast to resource availability prediction. GHS addresses the performance issues of computation and communication. It is designed to integrate novel stochastic and analytical modeling with newly developed scheduling and rescheduling methodology to utilize the performance, enhance Quality of Service (QoS), reliability, and trust of Grid Computing.

2. Grid Harvest Service

A series of technical challenges arise in Grid computing due to resource availability and heterogeneity. This includes evaluating resource availability on application performance, partition and schedule a parallel application accordingly, and the support of dynamic scheduling. The Grid Harvest Service system comprises of five primary subsystems: performance evaluation, performance measurement, task allocation, task scheduling, and execution management. Coordinately, they provide appropriate services to harvest Grid computing.

2.1. Performance evaluation

The most challenging technical hurdle of task scheduling in shared environments is to estimate the resource availability and to find its influence on the application performance. Analytical modeling in general has limited success in capturing the effect of "sharing" of non-dedicated resources. Nonetheless probability and stochastic modeling are often either too simplified or cannot reach a meaningful solution for a given

engineering application. Existing models do not match the complexity of the Grid.

Computation modeling. We have developed a model for non-dedicated computing [GoSW02]. It was derived from a combination of rigorous mathematical analysis and intensive simulation to make it generic and practically useful. The model considers the heterogeneous machine utilization and computing capacity, heterogeneous job arrival rate as well as heterogeneous service distributions. The effects of machine utilization, computing power, local job service and task allocation on the completion time of remote task are individually identified.

We refer the application under scheduling the remote task and the other competing local processes the local jobs. As observed and reported by researchers at Wisconsin-Madison, Berkeley, Maryland, et al, the arrival of local jobs in machine usage patterns follows a Poisson distribution with λ, the service time of local jobs follows a general distribution with mean $1/\mu$ and standard deviation σ. The cumulative distribution function of the remote task completion time is derived as:

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

where $\rho = \lambda/\mu$ is the machine utilization, $\theta = \sigma\mu$ is the coefficient of variation of service, τ is the computing capacity, and w is the workload of the remote task. The first term on the right-side of equation (1) is the performance without interruption. The second term, $(1 - e^{-\lambda w/\tau}) \Pr(U(S) \le t - w/\tau \mid S > 0)$, is the performance with the interruptions. Special efforts were made to use intensive simulations to find the distribution of the second term so that equation (1) can be used in actual performance predication. In the case of a parallel application, $\Pr(T \le t) = \prod \Pr(T_k \le t)$ is used to calculate the cumulative distribution function of the application completion time. Initial experimental results confirm the theoretical finding and show this model is practical and works well.

Communication modeling. Modeling of end-to-end network performance is essential for estimating the communication cost of message transfer between parallel processes and data transfer among dependent tasks. Queuing theory has been widely applied in the analysis of network performance based on the assumption of Poisson job arrival with exponential service time. However, this assumption is against the observation of real-world network traffic, which presents self-similar properties. Moreover, the end-to-end network path between two remote nodes may dynamically change to adapt the variation of network traffic. These characteristics make network performance modeling extremely challenging. Existing models do not capture the complicated short and long-range temporal dependence characteristic of wide-area network traffic.

We have applied ANN (Artificial Neural Network) techniques to model network traffic [EsSW05]. The strengths of ANN are its outstanding learning abilities, robustness to noise, and need of little prior knowledge. By online learning, ANN model can take into account the changes in the environmental conditions and adapt itself to the changes. Our ANN model predicts network performance in terms of available bandwidth and latency, and consists of five basic steps: collecting network traffic data; setting the right bin size and prediction step; preparing input parameters for neural network training; choosing appropriate ANN parameters such as learning rate, epoch number, and layer structure; and verification of established ANN model. A challenge in the construction of neural network model is the tradeoff between prediction accuracy and cost. After exhaustively examining all combinations of possible input parameters derived from the real-world network traffic trace files provided by WAND, ITA, and MOAT [EsSW05], we have identified the following parameters as most useful information for bandwidth prediction: timestamp, average packet rate, average bit rate, and their past information. To further improve the prediction accuracy, we examine the network traffic composition in the trace files. We find the network traffic in trace file, which is usually composed of different types of application traffic like TCP, UDP, ICMP and others. The application traffics statistics indicate that each type of traffic data presents a different traffic pattern. Instead of training one neural network to learn different patterns, we can construct an individual neural network model for each of them and then combine the individual prediction results into the overall traffic prediction.

2.2. Task allocation

Similar to parallel computing, Grid computing generally involves three steps: task allocation, task scheduling, and task execution. Task allocation decides how to partition an application into subtasks and then task scheduling maps them to a chosen set of resources for optimal performance. Workload balance approach is widely used in parallel computing. Conventional parallel computing tools cannot directly be applied to the Grid due to its heterogeneous and dynamic resource availability and capacity. A mean-time task partition algorithm is thus developed in GHS 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. A min-min algorithm is implemented to cluster subtasks of a meta-task and map each set of subtasks to a resource based on the prediction of the execution time of each set. The basic idea of these two algorithms is execution time balance, in contrast of the conventional

workload balance approach. A detailed description of task partition with respects of CPU, memory, network resource heterogeneity and resource sharing is given in [WuSu04].

2.3. Task scheduling

The GHS task scheduling takes the prediction from the performance evaluation subsystem. It supports different scheduling scenarios according to the application's requirement. Scheduling algorithms are proposed and tested for sequential task, parallel program, and meta-task, respectively. A heuristic scheduling algorithm is proposed to find a near optimal solution with a reasonable cost based on our computing model [SuWu03].

A challenge of task scheduling in Grid computing is 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. To provide reliable scheduling, we implement a trigger system that uses the GHS performance measurement subsystem to collect information and automatically trigger rescheduling [DGSS04]. The trigger system is rule-based and can add in new abnormal conditions. A task-rescheduling algorithm and associated environment is developed [WuSu04]. The subtasks on resources showing abnormal performance are assigned to other appropriate resources based on a re-estimation of the application completion time. A variant of formula (1) is used to calculate the cumulative distribution function of the application execution time in this situation.

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

where $\Pr(T_i \le t) = e^{-\lambda_i w_i'/\tau_i} + (1 - e^{-\lambda_i w_i'/\tau_i}) \Pr(U(S_i) \le t - t_i - w_i'/\tau_i \mid S_i > 0)$. t_i denotes the execution time of subtasks on machine i by so far and w_i' denotes the workload of subtasks on machine i that haven't been completed. $w_{\max} = \max\{t_i + w_i'\}$.

2.4. Execution Management

Execution management subsystem carries the task partition, scheduling, and re-scheduling policies. It consists of two parts: task management and execution engine. The task management maintains the map of application's subtasks among resources and their running status. The execution engine serves job submission, monitoring, and task re-scheduling. It also cooperates with the performance evaluation subsystem to identify abnormalities. When either a subtask is finished or a performance abnormality is identified, the execution engine reports it to the task management component. In the latter case, a task re-scheduling will be triggered and task re-scheduling algorithms will be invoked to find where and when the subtasks should be moved or migrated. Funded by the NSF Middleware Initiative (NMI) program, we have developed a process migration system, named the High Performance Computing Mobility (HPCM) [HPCM03, DuSC03], which supports run-time process migration of native codes written in C or Fortran in a heterogeneous Grid environment between different virtual organizations. HPCM provides the infrastructure for actual run-time dynamic scheduling and task reallocation.

3. Software Architecture

The Grid Harvest Service system is designed based on the novel prediction, partition, and scheduling mechanisms discussed above.. Its software architecture is depicted in Figure 1. Its major components include performance measurement engines, system-level predictors, an application-level predictor, a task allocator, a task scheduler, execution engines, as well as a task manager. The performance evaluation subsystem is designed with an application-level predictor and system-level predictors component that are distributed on each resource. The execution management subsystem consists of a task manager and an execution engines component.

The life cycle of a meta-task scheduling with GHS is presented as follows to illustrate how each GHS component collaborates to enable the running of applications in shared environments.

1. A user uses a task editor to compose the meta-task he wants to submit. He can select a group of machines for his application running or leaving it to the task manager. The task manager will collect the resource information and decide which set of machines is appropriate.
2. The task manager sends a task scheduling request to the task scheduler, which includes the application information and resource information. Based on the request, the task scheduler consults the system-level predictor on correspondent resources to get the estimated resource availability. The task scheduler then contacts the task allocator and the application-level predictor to find an optimal plan that satisfies the user's requirement.
3. The task manager records the mapping information of subtasks among resources returned by the task scheduler. After that, the task manager sends subtasks and their input and output files to the execution engines on correspondent resources according to the map information through *scp* or the GridFTP service.
4. After receiving the subtask allocation information, the execution engine (EE) consults the local resource management system to submit application's subtasks. It monitors the subtask running and maintains the status of subtasks. When a subtask is completed, the execution engine sends a message to update the application status in the task manager.

5. During the execution of subtasks, the performance measurement engine (PME) monitors the resource status and the execution engine collect the application running information. The execution engine periodically compares the observed resource behavior and the expected resource behavior obtained through the system-level predictor (SLP). If abnormal situations are detected, a rescheduling request is sent to the task manager.

6. The task manager collects the latest status of tasks and resources in the system and sends the information to the task scheduler for task reallocation. Based on the generated rescheduling plan, the HPCM is invoked to move the application's subtasks from abnormal resources to appropriate machines.

Step 4 – Step 6 are repeated until the completion of a Grid meta-task.

The above software architecture allows seamless integration of GHS components with Grid services to enable efficient performance evaluation and task scheduling. The Grid Information Service can be used by the task manager to locate potential available resources. The Grid FTP service can be used to handle the transfer of applications and their data files and the GRAM can be used in execution engines to dispatch subtasks on resources. Following OGSA, the task scheduler and the application predictor can be represented as Grid services so that they can serve other Grid Services, such as the Grid-enabled Programming System (GEPS), Problem Solving Environment (PSE), in a Grid runtime system.

4. Experimental results

We have partially implemented the GHS system as a proof of concept. Initial experimental results, collected at the Argonne and Oak Ridge national laboratories, as well as at IIT, are very encouraging. They confirm the GHS design principle and its potential. We have compared the prediction error of GHS and NWS [WoSH99] and the performance of GHS scheduling and AppLeS [BWCC03] scheduling for long-term applications [SuWu03, EsSW05]. We consider NWS and AppLeS, as they are the best-used performance prediction and task scheduling system, respectively, in current Grid computing practice.

Performance Evaluation. The Network Weather Service [WoSH99] provides short-term system-level performance prediction based on various simple forecasting methods. As it claims, it is suitable for jobs of five minutes time span or less. To evaluate the accuracy of the prediction model, we define the prediction error as $\left|\frac{Prediction_{period} - Measurement}{Measurement}\right|$. Figure 2 shows an experiment conducted on the 64-node Sun ComputeFarm, named *Sunwulf*, at IIT. The application is a replication of NAS Serial Benchmarks. The class type of these benchmarks is "A" or "W". The local job's lifetime is simulated with the observation of real-life processes

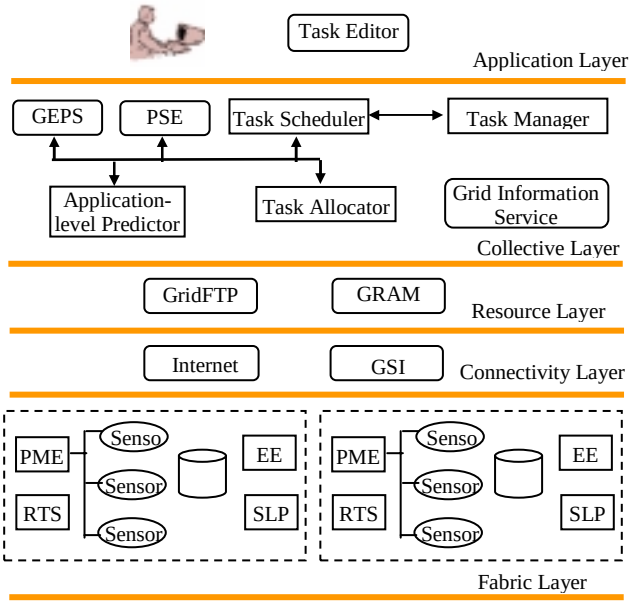


Figure 1. Software structure of Grid Harvest System

[SuWu03]. The three predictions are based on NWS prediction in terms of 10 seconds (default set of NWS) and 5 minutes, and prediction provided by GHS, respectively. It shows that the prediction error based on NWS remains very high while the prediction error based on GHS decreases with the increase in application workload. This comparison shows that the GHS approach is fundamentally more appropriate for long-term applications.

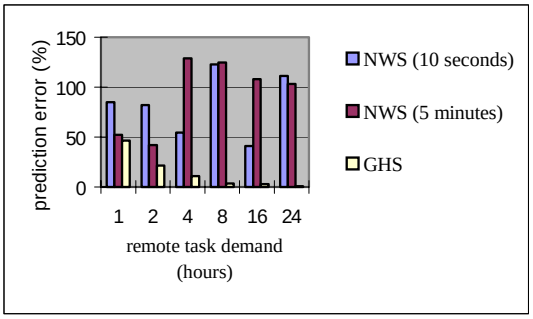
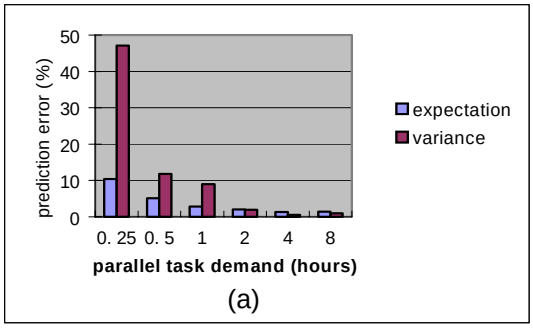
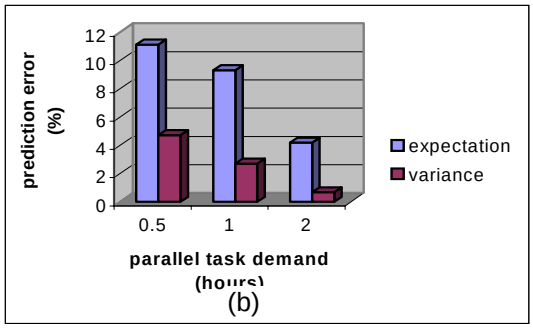


Figure 2. Mean of the prediction error of NWS and GHS

Figure 3 (a) gives the expectation and variance of the prediction error on the parallel program completion time with different task demands (from 4 to 256 hours sequential processing time) on 32 nodes of the *Sunwulf*. Each node is simulated with different usage patterns. The expectation and variance of the prediction error get smaller with the increase in job length. We also evaluate the prediction model on actual Grid environments. Figure 3 (b) shows the prediction error of a remote parallel task completion time on *Pitcairn*, a productive Grid node at Argonne National Laboratory. *Pitcairn* is a



(a)



(b)

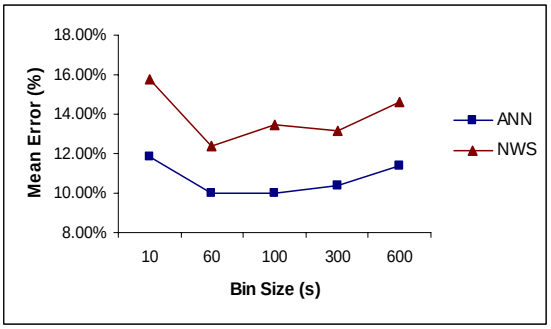
Figure 3. Expectation and variance of prediction error on parallel machines

multiprocessor with 8 250MHz UltrasparcII processors and 1GB of shared memory. It is a Grid node shared by many users. The result again shows that the expectation and variance of the prediction error get smaller as the demand of remote task increases.

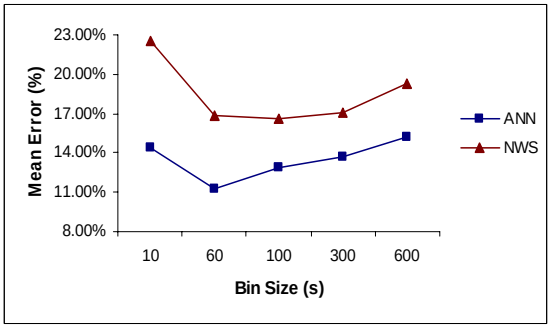
To verify the efficiency of the proposed Neural Network based prediction approach for network performance estimation, comparison is made with that of NWS. Figure 4 shows the performance comparisons of ANN and NWS for varying bin sizes for one-step prediction of AUCKLAND IV and II traces [NLAN04]. From the graph we can see that the prediction results of ANN supersede those of NWS for each bin size, illustrating the performance of the ANN mechanism is noticeably better than that of NWS. Compared with the prediction error of NWS, the performance gain of ANN prediction is 26.1% for AUCKLAND IV and 34.4% for AUCKLAND II. ANN is even more powerful for a given network application due its ability to learn [EsSW05].

Task scheduling. AppLeS makes scheduling decision through estimating the mean of subtask execution time based on the prediction of resource utilization provided by NWS. Other system specific factors are not analyzed due to the inherent limitation of these methods. The GHS task scheduling system provides long-term application-level performance prediction. The effects of machine utilization, computing power, local job service, and parallel processing on the completion time of parallel task are considered.

We choose the parameter sweep application to compare AppLeS and GHS because AppLeS only



(a) AUCKLAND IV



(b) AUCKLAND II

Figure 4. Performance comparison of ANN and NWS for one step prediction

supports meta-task scheduling but not general parallel processing. Since we have proven NWS does not work for long-term predictions, for a fair comparison, we have modified AppLeS to let it access GHS' prediction. The comparison of application completion time (seconds) and the number of machine set with the two different scheduling systems in a simulated Grid environment is given in Table 1. Simulation results show that the application completion time with GHS is 10%-20% less compared with that of AppLeS while only uses about one-half of the number of machines used by AppLeS. The GHS scheduling uses fewer machines and finishes in a shorter time than AppLeS for large applications. The reason is that GHS scheduling considers the effect of machine availabilities on parallel processing while AppLeS does not. GHS scheduling is designed for shared, dynamic systems. It has a real potential.

Table 1. A 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. Related work

Performance evaluation techniques have been widely used in parallel and distributed programming environments. Some well-known systems include Paradyn, TAU, Prophesy, and SCALEA [SuWu03]. These performance evaluation tools measure and analyze the application performance. However, they focus on application performance in a dedicated parallel system instead of a non-dedicated distributed environment. They don't provide performance prediction based on resource availability. The NWS [WoSH99] monitors and forecasts resource performance on-line. RPS Toolkit [DiHa99] predicts the CPU availability of a Unix system over a small time range with the time series techniques. These works are for non-dedicated environments. However, they only predict the short-term (five minutes or less, with a good prediction around 30 seconds) availability of non-dedicated resources. There is no application-level performance analysis and long-term prediction.

Most scheduling methodologies in distributed systems concern on application performance or system load balance issues. They are based on either current system usage or advanced resource reservation mechanism. Reservation asks resource owners giving up their privilege and may suffer in system utilization. It might be useful for high priority tasks or to show the potential of Grid computing, but has difficulties to be fully employed in a general enterprise environment. The experience in the development of the GrADS project [BCCD01] and other Grid projects has demonstrated that the integration of performance evaluation mechanism with application is pivotal to the success of Grid environments.

6. Conclusions

We have presented the mechanisms and design of the GHS system for Grid performance prediction and task scheduling. A prototype of GHS is under development, which consists of performance evaluation, performance measurement, task allocation, task scheduling, and execution management subsystems. Initial experimental testing is conducted on production machines at Argonne National Laboratory, Oak Ridge National Laboratory, and IIT and on real wide-area network traffic collected by WAND, ITA, and MOAT. Experimental results show that GHS adequately captures the dynamic nature of Grid computing. The performance gain of GHS prediction significantly supersedes other prediction systems such as NWS. Experimental results also show that, in scheduling of large applications on a non-dedicated heterogeneous environment, GHS scheduling decreases the task completion time by 10%-20% than that of AppLeS, while using only about one-half of the machines used by AppLeS.

Current GHS implementation separates the computing consideration with communication consideration, and is only for the proof of concept. Supported by the NSF NGS program, we are working to fully implement the GHS program. We plan to release a prototype GHS system for computing intensive applications in 2005. With continued funding, we plan to extend the computing-only GHS to support communication intensive application as well, and embedded GHS into Grid environment seamlessly via Java CoG Kit as Grid service.

Acknowledgments

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

References

[BCCD01] F. Berman, A. Chien, K. Cooper, J. Dongarra, et al, "The GrADS Project: Software Support for High-Level Grid Application Development", International Journal of High Performance Computing Applications, Vol. 15, No. 4, pp. 327-344, 2001.

[BWCC03] 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.

[DGSS04] C. Du, S. Ghosh, S. Shankar, and X.-H. Sun, "A Runtime System for Autonomic Rescheduling of MPI Programs," in the Proc. of the 33rd International Conf. of Parallel Processing, Montreal, Canada, August 2004.

[DiHa99] 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.

[DuSC03] 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.

[EsSW05] A. Eswaradass, X.-H.Sun, M. Wu, "A Neural Network Based Predictive System for Available Bandwidth," accepted to IPDPS2005, 2005.

[FoKe04] I. Foster and C. Kesselman, The Grid2: Blueprint for a New Computing Infrastructure, Morgan-Kaufman, 2004.

[GoSW02] 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.

[HPCM03] HPCM: High Performance Computing Mobility, http://www.nsf-middleware.org/NMIR4/contrib/download.asp.

[SuWu03] X.-H. Sun and M. Wu, "Grid Harvest Service: A System for Long-Term, Application-Level Task Scheduling," in Proc. of 2003 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003), Nice, France, April, 2003.

[WoSH99] R. Wolski, N. T. Spring, J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," J. Future Generation Computing Systems, Vol. 15, No. 5-6, pp. 757-768, 1999.

[WuSu04] 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.