Grid Harvest Service: A System for Long-term, Application-level Task Scheduling

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

Abstract

With the emergence of grid computing environment, performance measurement, analysis and prediction of non-dedicated distributed systems have become increasingly important. In this study, we put forward a novel performance model for non-dedicated network computing. Based on this model, a performance prediction and task scheduling system called Grid Harvest Service (GHS), has been designed and implemented. GHS consists of a performance measurement component, a prediction component and a scheduling component. Different scheduling algorithms are proposed for different situations. Experimental results show that the GHS system provides satisfactory solution for performance prediction and scheduling of large applications and that GHS has a real potential.

Keywords: *performance prediction and measurement, task scheduling, resources sharing, grid computing, performance modeling*

1. Introduction

Performance evaluation has always been an important issue in computer science, especially in the field of high performance computing. Many factors, including computer architecture, network latency, compiler techniques and application algorithms, affect application performance in a high performance computing environment. In the early 1990's, inspired by the success of Internet technology, a pervasive computational environment composed of a large number of heterogeneous and dynamic network resources was conceived and constructed. While this new complex environment provides the potential computing power, it also introduces a big challenge in task allocation and scheduling [1]. How to partition and where to allocate tasks in such a large, available but shared system still elude the researchers. The key to reach an optimal scheduling in such an environment is performance prediction.

In this study, we present a performance prediction and task scheduling system, the Grid Harvest Service (GHS) system, which provides long-term application-level performance prediction based on a newly proposed performance model. Early work in performance modeling was mostly focused on dedicated systems. The study of usage patterns of non-dedicated workstations is relatively recent. Mutka and Livny [2] reported that the distribution of available time intervals on workstations could be characterized as combination of several hyper-exponential distributions. Harchol-Balter and Downey estimated the process execution time [3]. Based on their experimental observation, they claimed that the median remaining life of a process is equal to its current age. These work is observational in nature. Leutenegger and Sun [4] put forward an analytical performance model to investigate the effect of a remote task on the local jobs of the workstation owner and vice versa. An effective prediction formula was derived for homogeneous non-dedicated systems. Most recently, Gong, Sun, and Watson have introduced a more general model for heterogeneous non-dedicated network computing [5]. This model was derived from a combination of rigorous mathematical analysis and intensive simulation to make it generic and practically useful. The effects of machine utilization, computing power, local job service and task allocation on the completion time of remote task are individually identified. Formulas to distinguish the impact of different factors are derived in the model analysis, which provide us the guildline for performance optimization.

There are several on-going projects on performance evaluation in parallel or distributed programming environment. However, there is still no adequate solution for general enterprise network environments. Paradyn Parallel Performance Tools [6] 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. But it does

not provide performance analysis and prediction based on resource usage pattern. TAU (Tuning and Analysis Utilities) [7] 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 option of the library. It is a post-execution performance analysis system. These systems focus on application performance in a dedicated parallel system instead of a non-dedicated distributed environment. The Network Weather Service [8] monitors and forecasts resource performance on-line. It provides system performance sensors, various simple forecasting methods, dynamic predictor selection and web-based visualization interfaces. In RPS (Resource Prediction System) Toolkit [9], Dinda predicts the CPU availability of a Unix system over a small time range with the time series techniques. Their work is for non-dedicated environments. However, their work only predicts the availability of non-dedicated resources. There is no application-level performance analysis or prediction. Furthermore, their prediction focuses on short-term performance.

The experience in the development of the GrADS project and other grid projects has demonstrated that the integration of performance evaluation mechanism with application is pivotal to the success of grid environments [10]. Currently, the scheduling algorithm in the APPLES [11] project is supported by the short-term system prediction provided by NWS services. In this study, we present the prototype development of the GHS system for long-term application-level performance prediction and task scheduling. We discuss the modeling foundations, introduce the measurement mechanisms, derive scheduling schemes, and present initial experimental testing results. Analytical and empirical results show that the prototype GHS system provides satisfactory solution for performance prediction and task scheduling of grid computing and that it has real potential.

2. Performance modeling and analysis

Our system is based on the modeling results derived in [5]. The model assumes that the local tasks have high priorities. For the development of GHS, we have extended the results for equal priority competition. That is the distributed task has priority equal to local tasks in competing for resources. This extension is appropriate for a grid environment where different remote users may compete for resources under the same priority. The analysis of equal priority competition shows for large remote tasks, competing for resources gives a little gain for the remote task but leads to a noticeable impact on the local jobs.

To distinguish the grid task under scheduling with other user's competing jobs, we call the grid task the remote task and the other competing jobs the local (sequential) jobs. We assume that a grid task can be divided into independent sub-tasks for parallel processing and the arrival of the local jobs at machine k follows a Poisson distribution with λ_k. The service time of local jobs at machine k follows a general distribution with mean $1/\mu_k$ and standard deviation σ_k. Based on our assumption, the owner job process is a M/G/1 queuing system. These assumptions are used in [5] and are based on the observations of machine usage patterns reported by researchers in Wisconsin-Madison, Berkeley, Maryland and et al [12]. We assume that the remote task is composed of one single parallel phase and a final synchronization phase.

2.1. Completion time of a remote task

The remote task is given a lower priority than the local job so that the remote task is less intrusive. The total work demand of the remote task is W. Each machine k ($1 \le k \le m$) has a sub-task work w_k and speed τ_k. The completion time of the sub-task on machine k can be expressed as:

$$T_k = w_k / \tau_k + Y_{k1} + Y_{k2} + ... + Y_{kS_k} \quad (1)$$

$Y_{ki} (1 \le i \le S_k)$ is the computing time consumed by sequential jobs and S_k is the number of interruption due to local job arrivals on machine k. By defining

$$U(S_k) = \begin{cases} 0, & if \quad S_k = 0 \\ Y_{k1} + Y_{k2} + ... + Y_{kS_k}, & if \quad S_k > 0 \end{cases} \quad (2)$$

We can obtain the distribution of T_k as

$$\Pr(T \le t) = \begin{cases} e^{-\lambda_k w_k/\tau_k} + (1 - e^{-\lambda_k w_k/\tau_k}) \Pr(U(S_k) \le t - w_k/\tau_k \mid S_k > 0 \\ \qquad if \quad t \ge w_k/\tau_k \\ 0, \qquad otherwise \end{cases} \quad (3)$$

If the distribution of $\Pr(U(S_k) \le u \mid S_k > 0)$ can be identified, we can calculate the distribution of sub-task completion time. Using the well-known result in queuing theory, we can get the mean and variance of sub-task completion time [5]. The mean and variance of $U(S_k)$ given $S_k > 0$ are thus calculated as:

$$E(U(S_k) \mid S_k > 0) = \frac{1}{1 - e^{-\lambda w}} \frac{1}{1 - \rho} \frac{w_k}{\tau_k} \quad (4)$$

$$V(U(S_k) \mid S_k > 0) = \frac{1}{1 - e^{-\lambda_k w_k}} \frac{\rho_k}{(1 - \rho_k)^3} \frac{(\theta_k^2 + 1)}{\mu_k} \frac{w_k}{\tau_k} \quad (5)$$

where $\rho_k = \lambda_k / \mu_k$ is the machine utilization and $\theta_k = \sigma_k \mu_k$ is the coefficient of variation of service.

The completion time of a remote task is the maximum of each sub-task completion time. After the distribution of the completion time of sub-task w_k is identified, the cumulative distribution function of the remote parallel task completion time can be calculated as:

$$\Pr(T \le t) = \begin{cases} \prod_{k=1}^{m} [e^{-\lambda_k w_k/\tau_k} + (1-e^{-\lambda_k w_k/\tau_k}) \Pr(U(S_k) \le t - w_k/\tau_k \mid S_k > 0)] & \\ \qquad if \quad t \ge w_{\max} & \\ 0, \qquad otherwise & \end{cases} \quad (6)$$

where $w_{\max} = Max\{w_k / \tau_k\}$.

Simulation results indicate that Gamma, Lognormal or Weibull are among the best-fit distributions to describe the $\Pr(U(S_k) \le u \mid S_k > 0)$. When the machine utilization is less than 15%, the Gamma distribution is the best. The Weibull distribution favors $\Pr(U(S_k) \le u \mid S_k > 0)$ when machine utilization is medium. If the utilization is higher than 50%, the Lognormal distribution may be the best choice. In general, the Gamma distribution is appropriate for the calculation of $\Pr(U(S_k) \le u \mid S_k > 0)$.

By evaluating the mean and coefficient of the task completion time on a group of available machines, the task can be assigned to the most appropriate resources. How to define selection criteria with the mean and coefficient of variation is determined by application requirements. In GHS, we choose the machine set with the smallest $E(T)(1 + Coe.(T))$.

2.2. Remote task partition and allocation

If the sub-task demand w_k is given to machine *k*, we can calculate the mean and variance of the parallel completion time using formula (6). The question here is how to partition a task and allocate sub-tasks to machines so that we can achieve an optimal performance for a given number of machines. A natural strategy is that machine *k* will be assigned a sub-task w_k so that the mean sub-task completion time is the same at different machines. Suppose the mean sub-task completion time is α, we get the sub-task demand $w_k = \alpha(1-\rho_k)\tau_k$ since $E(T_k) = \frac{1}{1-\rho_k}\frac{w_k}{\tau_k}$. And because $W = \sum_{k=1}^{m} w_k$, the sub-task workload can be expressed as

$$w_k = \frac{W}{\sum_{k=1}^{m}(1-\rho_k)\tau_k}(1-\rho_k)\tau_k \quad (7)$$

By comparing the remote task completion times on different sets of machines with formula (6), we can identify the best set of machines for running the remote task.

3. The design of the Grid Harvest Service system

The general performance model has been verified via intensive simulation testing. The next question is how to apply the model in a general grid environment, how to measure the needed parameters in a general grid environment, how to measure them in a least intrusive way, and how to use the prediction for performance optimization. The measurement methodology and task allocation issues are discussed in this section.

3.1. Measurement methodology

According to the model, parameters λ_k, ρ_k, σ_k and τ_k should be measured in order to calculate the mean and coefficient of variance of the remote task completion time, where λ_k is the local job arrival rate on machine *k*, ρ_k is the machine utilization, σ_k is the standard deviation of service time and τ_k is the computing capacity of machine k. τ_k can be obtained by running computation intensive benchmarks. We focus on the measurement of λ_k, ρ_k and σ_k.

Suppose parameter x has a population with a mean and a standard deviation and we have a sample $\{x_1, x_2, ..., x_n\}$, the smallest sample size with a desired confidence interval and a required accuracy *r* is given by $n = (\frac{100\ z_{1-\alpha/2}\, d}{r\bar{x}})^2$ [13]. 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, then we get

$$n_s = 1536.64(\frac{d}{\bar{x}})^2 \quad (8)$$

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

$$d = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad (9)$$

In our experiment, we assume that parameter x is a random variable with a fixed mean and a fixed standard deviation during a continuous 24-hour period on each machine. Parameter x is measured in n_s time intervals during 24 hours. The average of x is viewed as a sample of x over 24 hours. A prediction of x for the next 24 hours is based on the history of sample x. Here we use a dynamic method to adjust the number of time intervals. $\bar{x}$ and s over the previous 24 hours are used to calculate n_s

with formula (8) at the end of each hour to adapt the possible variance of x. A number ($\frac{n_s}{24}$) of x_i will be measured for the next hour. The average of x taken at time t over previous 24 hours is calculated with the following formula:

$$Adapt\ \_avg\ (x,t) = \frac{1}{\sum_{i=t-23}^{t} |\ \mathrm{X}_i\ |} \sum_{i=t-23}^{t} \sum_{j=1}^{|\mathrm{X}_t|} x_{ij} \qquad (10)$$

where X_i means the set of x_{ij} measured in the ith hour. Now the problem is how to measure parameter x over a time interval $T_{interval}$.

The Unix utility *vmstat* is used to measure ρ_k. It accesses performance statistical data, which is collected and maintained by the kernel system. The system resources occupied by *vmstat* are negligible. In our experiment, we take each process as a job. The Unix utilities *ps* and *lastcomm* are used to obtain process execution information at the beginning and the ending of $T_{interval}$ to calculate ρ_k. *ps* shows the active processes information while *lastcomm* presents the previous executed processes information. We define J_{t1} as processes existing in the beginning of $T_{interval}$, $J_{between}$ as processes started and finished in $T_{interval}$, $J_{arrival}$ as processes started but not finished in $T_{interval}$ and J_{t2} as processes existing in the end of $T_{interval}$. So we get:

$$\lambda_i = \frac{J_{arrival}}{T_{interval}} = \frac{J_{between} + J_{start}}{T_{interval}}$$

$J_{between}$ will be given by comparing the output of *lastcomm* utility in the beginning with that at the end of $T_{interval}$. Since $J_{start}=\{J_i | J_i \in J_{t2}\ \ but\ \ J_i \notin J_{t1}\}$, we can identify J_{start} by looking into J_{t1} and J_{t2}.

To get σ_i, the standard deviation of service time, we need to measure each service time. During each time interval, we use *lastcomm* utility to get the executed processes service time and then calculate the average as a sample of service rate.

In our experiment, we calculate the number of measurements for the next hour according to the system history over the previous 24 hours. This method can dynamically adjust the measurement number to reduce the measurement cost. The GHS measurement system consumes very little CPU resource (less than 1%).

3.2. System integration and task scheduling

After parameters λ_k, ρ_k, σ_k and τ_k of each machine are measured and estimated, the completion time of a remote task can be predicted by using formula (6). The relation between the major components of the GHS system (shaded areas) and other grid services is shown in Figure 1. The task manager, which is responsible for task management, is located in the Application layer. It sends a request to the Scheduling component in the Collective layer for resource allocation. The Scheduling component contacts the Directory Service (DS) to locate the potential available resources. It then executes the task scheduling algorithm to identify the best set of resources by sending possible choices to the Prediction component and collecting the evaluation results. The Prediction component can also serve the grid-enabled programming systems and workload management systems in a grid runtime system. The Prediction component accesses the performance data to estimate the task completion time. The Performance Communication Manager (PCM) component is used to collect performance data, which is exchanged through the proposed performance data protocol (PDP) based on the communication mechanism provided by the GSI service in the Connectivity layer. The Performance Data Manager (PDM) component on each resource is responsible for measuring system and application information by using various sensors.

As shown in Figure 1, in addition to prediction, scheduling is also a primary component of GHS. A partition schema, called "equal-mean" partition, is given in equation (7). A set of scheduling algorithms have been derived and used in GHS. Figure 2 gives the scheduling algorithm for optimal parallel processing. If we have *m* idle machines, for the optimal algorithm, we need to check 2^m possible solutions. This is too costly when *m* is large. . A heuristic task scheduling algorithm is given in Figure 3 to find an acceptable solution with a reasonable cost. The basic idea is that machines with higher $(1-\rho_k)\tau_k$ are selected with higher priority. In Figure 3, w is the grid task demand, μ' is the average of the mean demand of local machines' tasks. Leutenegger and Sun [4] show that the task ratio, the ratio of the remote task demand to the mean demand of machine's local tasks, should be large enough to achieve acceptable efficiency. Here we choose it to be at least 4.

Figure 2 is a scheduling algorithm for parallel processing. Scheduling of parallel task is considered more challenging in a grid environment. Current scheduling algorithms used in grid environment are min-min [11] based algorithms for multiple independent remote tasks. Though not listed here, with $E(T_k)(1+Coe.(T_k))$ as the evaluation criteria, the optimal parallel processing scheduling algorithm can be extended for multi-independent task scheduling.

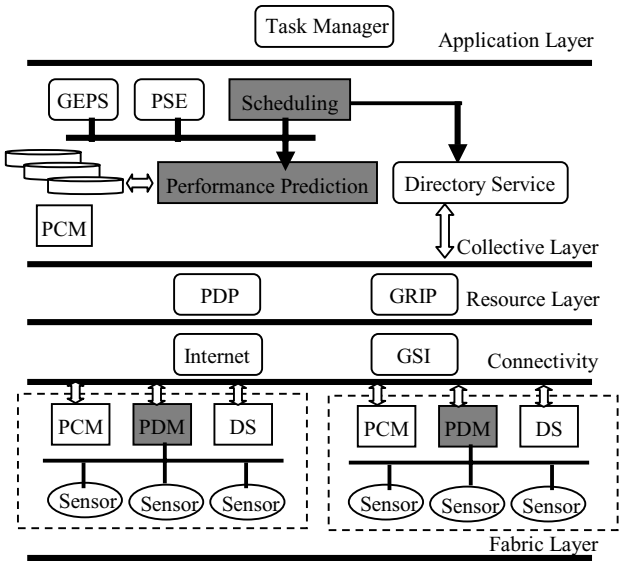

Figure.1 Integration of GHS components with other grid services

Assumption: a grid task can be partitioned into any number of sub-tasks. Each sub-task will be assigned to a machine respectively.
Objective: Scheduling a grid task with an optimal partition and allocation
Begin
List a set of idle machines that are lightly loaded over an observed time period, $M = \{m_1, m_2, \ldots, m_q\}$;

$p' = 1,\ k' = 1$;

$p = 1$;

While $p < q$

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

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

Use the formula (4) to partition sub-tasks to each machine in S_k^p;

Use the formula (3) to calculate $E(T_{S_k^p})(1 + Coe.(T_{S_k^p}))$.

If $E(T_{S_{k'}^{p'}})(1 + Coe.(T_{S_{k'}^{p'}})) > E(T_{S_k^p})(1 + Coe.(T_{S_k^p}))$

then $p' = p$; $k' = k$;

End If

End For

$p = p + 1$;

End While

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

End

Figure 2. Optimal task scheduling algorithm

Assumption: a grid task can be partitioned into any size of sub-tasks. Each sub-task will be assigned to a machine respectively.
Objective: Scheduling a grid task heuristically to reach a semi-optimal performance
Begin
List a set of idle machines that are lightly loaded over an observed time period, $M = \{m_1, m_2, ..., m_q\}$;

Sort the list of idle machines in a decreasing order with $(1-\rho_k)\tau_k$, $M' = \{c_1, c_2, ..., c_q\}$;

$a = 1, b = \min\{|M'|, \frac{w}{4 * 1/\mu'}\}$;

Repeat

$c = \lfloor (a+b)/2 \rfloor$

/* $f(x)$ denotes $E(T_{C(x)})(1 + Coe.(T_{C(x)}))$ where $C(x) = \{c_1, c_2, ..., c_x\}$ */

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

Figure 3. Heuristic task scheduling algorithm

4. 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 at IIT. The prediction precision of our analytical model is examined in both a simulation environment and an actual grid environment to verify the accuracy and feasibility of the GHS system. We have evaluated three allocation methods on a network of machines to examine our task partition strategy. The completion time of a remote task over different number of machines with different scheduling methods are compared. Finally, we investigate the relationship between the number of intervals measured in each hour and the variance of system utilization to test the efficiency of our dynamic measuring methodology in reducing the measurement cost. In our experiment, we choose NAS Serial Benchmarks (BT, CG, LU, MG, IS and SP) as the remote task. The class type of these benchmarks is "A" or "W".

4.1. Prediction error

To evaluate the accuracy of our prediction model, we define our prediction error as $|\frac{\Pr ediction_{period} - Measuremen\ t}{Measuremen\ t}|$. Figure 4 shows the expectation and variance of the prediction error of the remote task completion time with different job lengths (from 0.5 to 8 hours) on a Sun workstation. The workload is simulated based on observations from the SDSC Paragon logs and the CTC SP2 logs [14]. We can see that with increase in job length, the expectation and variance of the prediction error get smaller. When a remote task's workload is more than 8 hours, the expectation of the predication error of the task run time is less than 10%. We also investigated the prediction error for completion time of remote parallel task.

We conducted our experiment six times in a Sun ComputFarm cluster, named Sunwulf, at IIT. The expectation and variance of the prediction error on the parallel task completion time with different task demands (from 4 to 256 hours sequential processing time) on 32 nodes of the Sunwulf are given in Figure 5. We find that the prediction error reduces 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.

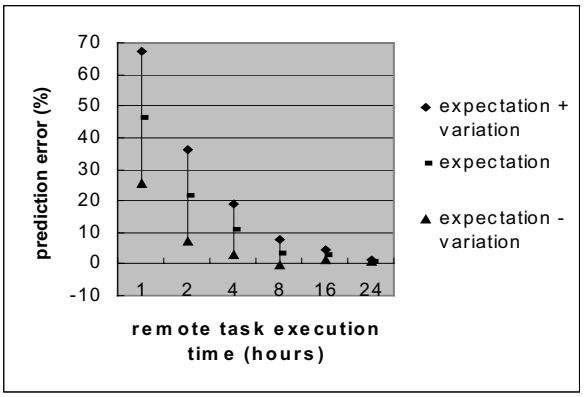

Figure 4. Expectation and variance of prediction error of remote task completion time on single machine

We have evaluated our prediction model on an actual grid environment with a practical workload. Figure 5 shows the expectation and variance of the prediction error of a remote parallel task completion time on *pitcairn*, a productive machine at Argonne National Laboratory. *Pitcairn* is a 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. When the demand of remote parallel task is 16 hours, i.e. average two hours workload is for each processor. The expectation of the prediction error is about 4.18%. The prediction error with 8 hours remote task demand is about 9.31%. Also we find that the prediction error reduces more quickly than that on a single workstation. Our experiment shows that our method can work even better on a virtual organization, which has its own local schedulers.

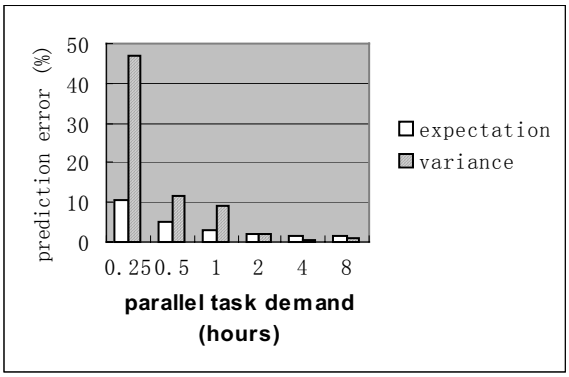

Figure 5. Expectation and variance of prediction error of remote task completion time on parallel machine

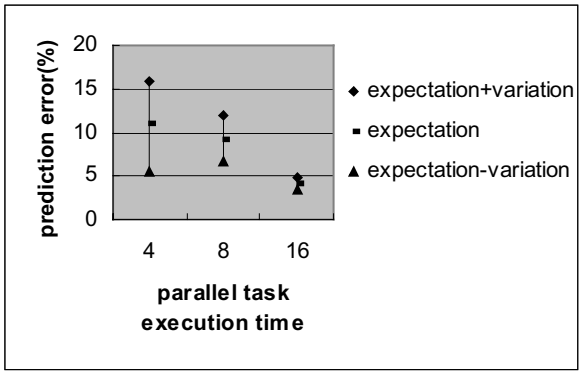

Figure 6. Expectation and variance of prediction error on multiple-cpu machine

4.2. Task partition and scheduling

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

We tested the efficiency of these partition approaches in two workstations. Workstation A has an average utilization of 50% while workstation B has an average utilization of 20%. The local jobs arrive with a Poisson distribution and are served with a Log-uniform distribution. The machines are with a speed ratio of 1.33:1. Figure 7 shows the remote task completion time with these

three partition approaches on two machines. The parallel workload is increased from 1 hour to 8 hours. Result shows that the mean-time partition is the best. The time saved by the mean-time allocation algorithm is 20%-25% for large jobs. The difference is significant. The detailed completion time of the parallel sub-task on machine A and machine B respectively show that the difference between the parallel sub-task completion time on machine A and machine B is the least with mean-time partition method. So the remote task completion time, the maximum of each sub-task completion time, is the least for mean-time allocation.

The task partition algorithm is used to identify how much workload of a grid task is 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 partition is another factor distinguishing GHS from other existing grid scheduling systems where partition is either not considered or equal-load partition is used. In section 3.2, we have discussed various scheduling algorithms. The heuristic task scheduling is proposed because of the high computing cost of the optimal task scheduling algorithms.

We conducted experiments to compare the performance of the two scheduling algorithms, optimal and heuristic task scheduling discussed in Section 3.2 in Sunwulf. The workload is simulated based on tracefiles from the SDSC Paragon logs and the CTC SP2 log [14]. The system parameters such as utilization, job arrival rate and service rate were varied at each node. The experiment was 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 selected a subset of machines for task allocation in random. The average run times of a remote task with different scheduling algorithms were compared. Our results demonstrate that the run time of the remote task and the number of utilized machines of heuristic task scheduling are close to those of optimal task scheduling. When scheduling task among 20 available machines, 14 machines were identified for optimal scheduling and 13 machines were used for heuristic scheduling. The average run time is 464.9 seconds for optimal scheduling and 486.4 seconds for heuristic scheduling. However, the computing cost of optimal task scheduling is increased from 3.16 seconds to 6558.75 seconds while the computing cost of heuristic task scheduling increased from 0.07 seconds to 0.25 seconds. When the number of available machines was 15, 11 and 9 machines were used for optimal scheduling and heuristic scheduling respectively. The same set of 8 machines was identified by optimal scheduling algorithm and heuristic algorithm when 10 machines were available. Table 1 shows the average run time of remote task with different scheduling strategies. R_n means a number of n machines are randomly selected for task allocation and scheduling and 20_m means that all of 20 machines are used in task scheduling.

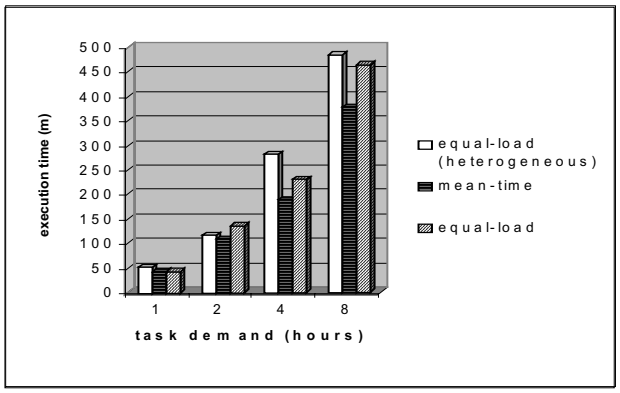

Figure 7. Comparison of the three partition approaches

Table 1. Average execution time of remote task with different scheduling strategies

M_N	R_5	R_{10}	R_{15}	20_m	Optimal	Heuristic
10	1587.9	901.2			792.4	792.4
15	1421.9	855.1	631.3		523.5	548.4
20	1329.5	798.4	619.2	600.4	464.9	486.1

4.3. Run-time cost

In our experiment, we calculate the number of measurement for the next hour according to the system history over the previous 24 hours. Our program can dynamically adjust the number of measurements to reduce the measurement cost. Figure 8 shows an example of the fluctuation of the number of intervals measured during 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 also measured the execution time of our prediction program. The experimental results show that the run-time cost of our prediction component is 0.66 seconds when the number of machines is 1024. Compared to the potential gain from task scheduling, the run-time cost is negligible.

5. Conclusion and future work

In this paper, performance prediction and task scheduling of large parallel or sequential tasks in a grid environment are studied. First, a new modeling result is identified and enhanced. Next, measurement methods and mechanisms are developed to measure the needed system parameters, and task partition and scheduling algorithms are introduced. A performance measurement and prediction system, the Grid Harvest Service (GHS) system, is then developed for grid computing. Finally,

initial experimental testing was conducted. Experimental results show that GHS adequately captures the dynamic nature of grid computing. For large jobs (eight hours sequential runtime or more), its prediction error is less than 10%. Its mean-time partition approach can reduce the computation time by 20% to 30% compared to partition without considering resource sharing. In addition, GHS is both non-intrusive and efficient. Its run-time cost is always less than 1%. Though our current experimental testing is preliminary, every indication shows GHS has a real potential in grid computing.

GHS is a long-term, application-level performance prediction and task scheduling tool for non-dedicated grid computing. It is a complement of existing 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 both short-term and long-term prediction. GHS can be combined with APPLES for general application-level 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. The current prototype implementation only demonstrates the feasibility and potential of the GHS approach. More work is needed to integrate GHS seamlessly into the grid system.

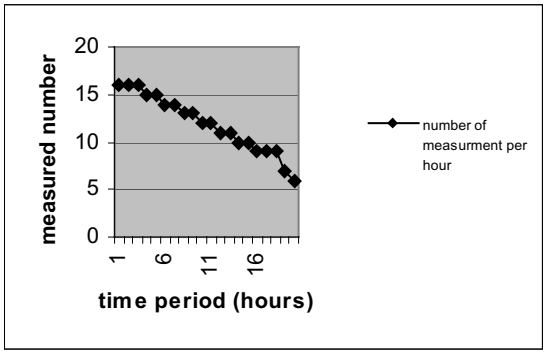

Figure 8. Decreasing of measurement when the system is steady

Acknowledgments

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

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.

References:

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, ISBN 1-55860-475-8, Morgan Kaufmann Publishers, San Francisco, CA, July 1998.

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

[3] M. Harchol-Balter and A. B. Downey, *Exploiting Process Lifetime Distributions for Dynamic Load Balancing*, Proc. SIGMETRICS, pages 13-24, May 1996.

[4] S. T. Leutenegger and X. H. Sun, *Limitations of Cycle Stealing of Parallel Processing on a Network of Homogeneous Machines*, Journal of Parallel and Distributed Computing, Vol. 43, No. 3, pages 169-178, June 1997.

[5] L. Gong, X. H. Sun, and E. F. Waston, *Performance Modeling and Prediction of Non-Dedicated Network Computing*, IEEE Trans. on Computer, Vol. 51, No 9, September, 2002.

[6] B. P. Miller and A. Tamches, *Fine-grained dynamic instrumentation of commodity operating system kernels*, Third Symposium on Operating Systems Design and Implementation (OSDI'99), New Orleans, pages 117-130, February 1999.

[7] S. Shende, A. D. Malony, J. Cuny, K. Lindlan, P. Beckman and S. Karmesin, *Portable Profiling and Tracing for Parallel Scientific Applications using C++*, Proceedings of SPDT'98: ACM SIGMETRICS Symposium on Parallel and Distributed Tools, pages 134-145, August 1998.

[8] R. Wolski, N. T. Spring, and J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*, Journal of Future Generation Computing Systems, Vol. 15, No. 5-6, pages 757-768, October, 1999.

[9] P. Dinda and D. O'Hallaron, *An Extensible Toolkit for Resource Prediction In Distributed Systems*, Technical Report CMU-CS-99-138, School of Computer Science, Carnegie Mellon University, July, 1999.

[10] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, *The GrADS Project: Software Support for High-Level Grid Application Development*, International Journal of High Performance Computing Applications, Vol. 15, No. 4, pages. 327-344, 2001.

[11] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, *The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid*, Proceedings of Super Computer 2000, November 2000.

[12] A. Acharya, G. Edjlali, and J. Saltz, *The Utility of Exploiting Idle Workstations for Parallel Computation*, Proc. SIGMETRICS, pages 225-236, June 1997.

[13] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiey & Sons, Inc, New York, 1992.

[14] A. B. Downey, *A Parallel Workload Model and Its Implications for Processor Allocation*, 6th Intl. Symp. High Performance Distributed Computing, pages 112-123, Aug 1997.