Quality of Service of Grid Computing: Resource Sharing

Xian-He Sun[†‡], Ming Wu[†]
Illinois Institute of Technology, Chicago Illinois 60616 USA[†]
Fermi National Accelerator Laboratory, Batavia, IL 60510-0500[‡]
{sun, wuming}@iit.edu

Abstract

Rapid advancement of communication technology has changed the landscape of computing. New models of computing, such as business-on-demand, Web services, peer-to-peer networks, and Grid computing have emerged to harness distributed computing and network resources to provide powerful services. The non-deterministic characteristic of the resource availability in these new computing platforms raises an outstanding challenge: how to support Quality of Service (QoS) to meet a user's demand? In this study, we conduct a thorough study of QoS of distributed computing, especially on Grid computing where the requirement of distributed sharing and coordination goes to the extreme. We start at QoS policies, and then focus on technical issues of the enforcement of the policies and performance optimization under each policy. This study provides a classification of existing software system based on their underlying policies, a systematic understanding of QoS, and a framework for QoS of Grid computing.

1. Introduction

With the advance of network technology, many new distributed computing models are being constructed to harness geographically distributed computing and communication resources, such as business-on-demand, Web services, peer-to-peer networks, and Grid computing [1]. Typical examples of these systems include WebSphere, Gnutella, Skype, Seti@home, Condor, PPLive (a P2P television network), and Globus [2]. The system size of these systems scales from hundreds of nodes to tens of thousands of nodes, and even more. In these systems, resources are shared and collaborated to provide services/functionalities such as online shopping, online telephony and television, teleimmersion, online control of scientific instrumentation, and resource pooling. Much effort is being made in the standardization of protocols and interface for service orchestration and resource collaboration in these environments. With the maturity of these systems, with more and more users, Quality of Service (QoS) of these newly-emerged computing platforms is becoming more and more important.

Quality may mean different things for different users under different environments. In general, quality is a nonfunctional character such as performance, cost, security, reliability, et al., or a combination of them. In a shared network environment like Grid, there are several new issues related to QoS support that do not arise in a single computer system. The first issue is the variation of resource availability. This variation may be due to resource contention, dynamic system configuration, software or hardware failures, and other factors beyond the control of a user. The uncertainty of resource availability has a big impact on application quality. The second issue is parallel processing. The total workload of a large scale application is often partitioned into smaller pieces, called subtasks. These subtasks are then allocated to resources in a distributed system to be processed concurrently. The challenge of parallel processing in a shared network environment lies on that the computing resources may be heterogeneous and have individual availability patterns. The third issue is non-centralized control. In a general Grid environment, the computing resources are autonomous. Local schedulers schedule local jobs and the Grid scheduler does not have the control of the local jobs. These new QoS issues make supporting QoS of Grid computing extremely challenge.

Because of these difficulties, in spite of recent intensive attention, a suitable and broadly applicable QoS solution has been elusive. This is especially true for Grid computing, where computing resources cross different virtual organizations and applications are topically large in scale, requiring distributed concurrent processing and frequent coordination. In this study, we call the tasks scheduled by the Grid scheduler Grid tasks and the other processes or jobs running in the Grid environment local jobs. We focus on studying the relation between the Grid tasks and the local jobs and investigate the influence of resource sharing on QoS. We assume that both the Grid system and the local systems can work effectively. The study is on the new QoS issues of Grid computing, especially on the interaction between the two sides, the Grid tasks and the local jobs. In recent years, some efforts have been made to address the issues of sharing. But most existing QoS support is built under certain tacit assumptions. They often support QoS of one side in the cost of sacrifice another without a good understanding of the QoS issues between the two sides. For instance, seti@home gives the priority to local jobs. The Grid task executes only when no local jobs are running. There is no QoS for Grid tasks. Resource reservation has been proposed for Grid computing [3]. Resource reservation reduces system utilization and may disturb the normal operation of local jobs. The question, then, is how much resources need to be reserved for how long and on which resources. Currently the decision is made based on the demand of the Grid task or let the local scheduler to make the decision. Without a better understanding of the impact of resource reservation on QoS an appropriate decision cannot be made.

A good QoS solution requires a comprehensive investigation of the complex interaction between system characteristics, such as resource sharing, non-centralized control, heterogeneity, and dynamics; and application characteristics, such as parallel processing, computation or communication, hard guarantee or soft guarantee. To provide a uniformed QoS solution for general Grid computing, we propose to develop a software solution through increasing the fundamental understanding of QoS of Grid computing. We specifically study the relation between the QoS of Grid tasks and the QoS of local jobs and investigate the influence of resource reservation on local jobs. We divide the fundamental understanding of QoS into two stages: policymaking and optimization mechanisms. Policymaking decides the QoS policy of resource sharing among Grid tasks and local jobs. Optimization mechanisms obtain an optimum QoS under each QoS policy. Optimization mechanisms in turn consist of advanced performance modeling, resources management, and scheduling algorithms. The interference of Grid tasks and local jobs are considered in modeling. Task schedulers choose the best set of resources to optimize the application QoS based on the information provided by performance modeling. Resource management collects application and resource information, enforces the QoS policy, and carries out the scheduling decisions. This top-down study not only leads to a better understanding of QoS between the Grid tasks and local jobs, it also gives a constructive solution.

2. QoS policy

Grid computing requires the coordination of distributed network resources to solve non-trivial applications. The distributed resources may belong to different virtual organizations, shared by Grid tasks and local jobs, and are often autonomic controlled. A Grid management system has no control over the usage pattern of the shared resources. The uncertainty of resources makes QoS of Grid computing hard to achieve. QoS is a known technical hurdle preventing a broader adoption of Grid computing. There has been no well-conducted QoS study to balance the need of Grid tasks and local jobs. Distributed systems, such as Condor, NetSolve, Nimrod, and Globus [2], support Grid computing and facilitate resource sharing and collaboration. These systems adopt different QoS policies, usually implicitly, and try to provide a satisfactory QoS under their adopted policies. These policies often support QoS for one side and sacrifice that of the other – they perform well for certain applications but do not provide a satisfactory solution for general Grid computing. For example, in SETI@home, Entropia, and Condor, a Grid task is allowed to run only when no keyboard or mouse activities are detected on local resources. This sharing policy ensures that Grid tasks do not interfere with the execution of local jobs, but the QoS of Grid tasks is at the mercy of local users. We call this cycle-stealing based QoS policy the best-effort service. In an opposite direction, in Legion and Globus, resources can be reserved for the execution of Grid tasks based on service-level agreements [3]. Service-level agreement is a great tool to enforce different QoS policies in a Grid environment. However, due to the lack of understanding of QoS, in current practice the resource reservation is only used to ensure the QoS requirements of Grid applications. The influence of resource reservations on local jobs is not considered. As a result, the QoS of local jobs could be severely degraded, which in turn could lead to significant reduction of the liability of the service-level agreement. In fact, recent research [4] has shown that local users often power cycle computers immediately if uncomfortable machine slowness is observed. We call this reservation based QoS policy the real-time service. Real-time service is a preferred policy for real-time Grid applications, but not others. Ideally, a set of policies should be introduced to balance the QoS need of both Grid tasks and local jobs and adjust the policies dynamically under certain criteria for different applications and computing environments. To lead a better understanding of QoS, we first need to understand the QoS policies.

Our study of QoS is based on the observation that Grid environments do not have a central control and the quality of service has to be supported on shared resources. Recognizing that the sharing has two sides, Grid tasks and local jobs, and the QoS concerns of the two sides are quite different, we specifically study the relationship between the QoS of Grid tasks and the QoS of local jobs and intend to develop a system solution to meet the need of both sides simultaneously, or at least balance the requirements. In practice, there are three resource sharing policies widely used in the Grid computing community.

- *The best-effort policy*. In best-effort policy, local jobs have higher priority than a Grid task. A Grid task is allowed to execute only when there is no local jobs running. During the execution of the Grid task, if any local jobs arrive, the Grid task is either suspended or terminated. In Grid computing, we are more interested in the first scenario because Grid tasks are usually stateful large-scale scientific applications. In practice, the keyboard and mouse activities are usually monitored to detect the local job running status. The best-effort policy is adopted in the Seti@home and Condor project.
- *The real-time policy*. In real-time policy, Grid tasks have a higher priority over local jobs. According to the Grid task QoS requirement, a certain computing resources (CPU, Memory, I/O, network) are reserved for the dedicated use of the Grid task. This reservation is fixed during the execution of the Grid task. The real-time policy is adopted in the GASA project [3]. It is often used for resource allocation in supercomputer centers, in which a number of nodes in a cluster system or in a group of cluster are reserved for the dedicated running of a scientific application. This policy is best for test the potential of Grid computing and within a virtual organization where a priority can be defined globally.
- *The competing policy*. In the competing policy, a Grid task competes for resources with local jobs. The good part of the competing policy is that every one is equal; no one has to sacrifice for another. The bad part, on the other hand, is there is no guaranteed QoS for any one. This policy is usually applied in public domains, such as campus computing environments. For example, Condor and AppLeS [5] project support this policy so that workstations in their systems are harvested to increase the computing throughput.

Besides the three existing QoS policy, we identify two new resource sharing policies which has the potential to be applied in a general distributed environment to enrich the different relationship between local jobs and Grid tasks in terms of their qualities.

- *The constrained policy*. In the constrained policy, the QoS of Grid tasks or local jobs has to be maintained at certain level. Meeting the predefined QoS is the first priority of the constrained policy. For instance, if the constrained is to maintain the local job QoS at certain level, then resources only can be reserved to Grid tasks when the reservation does not reduce the local job QoS below the QoS level. In the meantime, Grid scheduler should reserve as mush as resources possible for its gains as long as the constraint condition is not violated. This provides a compromised resource sharing policy where a certain part of a resource is dedicated to a Grid task under the condition that local job QoS is affected within a limitation.
- *The balanced policy*. In the balanced policy, both the quality of Grid tasks and the quality of local jobs are considered and measured with some given criteria. The balanced policy can be viewed as a further extension of the constrained policy. Instead of enforcing QoS constraint on one side, the balanced policy weighs the priorities of both sides. When there is no enough resources to support QoS, both sides will suffer together with predefined proportion.

The policy list is incomplete and is only an initial point. The balanced policy, for instance, may lead to different policies based on different tradeoffs and concerns; different policies may need to be combined and adopted at different time to support the QoS of a given application. The competing policy may have different QoS goals: optimizing the Grid task performance, minimizing the slowdown of local jobs, or some weighted combinations. In addition, the Grid tasks, or local jobs, themselves may have different QoS needs and should be treated separately..

3. Performance modeling

After a QoS policy is chosen, the next question is how to achieve an optimum quality under the given policy. Quality is often measured in terms of performance. Improving quality then becomes an optimization problem. For example, for the best-effort service, we should partition and schedule the Grid task in such a way to optimize its performance (QoS); for the real-time service we should schedule the Grid task appropriately to minimize the QoS degradation of local jobs. A more complex task is to optimize the QoS of both Grid tasks and local jobs under a balanced QoS policy. The optimization mechanisms in turn rely on advanced resource management, performance modeling, and task scheduling. The factors of computation, communication, and the interference of Grid tasks and local jobs should be considered in modeling the behavior of Grid and local tasks under each QoS policy. A task scheduler then can choose the best set of resources to optimize the application QoS based on the information provided by performance modeling. Notice that QoS modeling can be also used in choosing an appropriate QoS policy for a given application. Resource management collects application and resource information, enforces the QoS policy, and carries out the scheduling decisions. Here enforcing QoS includes the enforcement of resource reservation or other service-level agreements. Figure 1 presents the relationship of QoS policies, optimization goals, and optimization mechanisms in Grid computing. Due to the sake of page limitation, we only present some representative optimization issues of some selected QoS policies in this study.

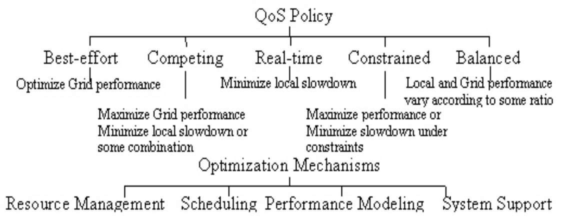
QoS Policy
Best-effort
Competing
Real-time
Constrained
Balanced
Optimize Grid performance
Minimize local slowdown
Local and Grid performance vary according to some ratio
Maximize Grid performance Minimize local slowdown or some combination
Maximize performance or Minimize slowdown under constraints
Optimization Mechanisms
Resource Management
Scheduling
Performance Modeling
System Support

Figure 1. QoS policy and optimization mechanisms

Computation modeling for best-effort service. We have introduced a performance modeling for best-effort service. The model predicts QoS, in terms of completion time, of a Grid task [6], so a set of appropriate resources can be chosen for optimal execution time. It is derived from a combination of rigorous stochastic analysis and intensive simulation and designed for large-scale applications. 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, and local job service and task allocation on the completion time of Grid task are individually identified.

In the best-effort service, the Grid task is given a lower priority than the local job so that the Grid task is less intrusive. We suppose the execution of the Grid task is interrupted by local jobs S times. Each busy period for local jobs running is $Y_i \quad (1 \le i \le S)$. The application completion time can be expressed as:

$$T = w/\tau + Y_1 + Y_2 + ... + Y_{S_k} \qquad (1)$$

$Y_i (1 \le i \le S_k)$ 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

$$U(S_k) = \begin{cases} 0, & if \quad S_k = 0 \\ Y_1 + Y_2 + ... + Y_{S_k}, & if \quad S_k > 0 \end{cases},$$

we can obtain the distribution of T_k as

$$\Pr(T \le t) = \begin{cases} e^{-\lambda w/\tau} + (1 - e^{-\lambda w/\tau}) \Pr(U(S_k) \le t - w/\tau | S_k > 0 & if \quad t \ge w/\tau \\ 0, & otherwise \end{cases} \quad (2)$$

where the distribution of $\Pr(U(S_k))$ can be appropriated with Gamma distribution [6]. In parallel processing, the completion time of a Grid 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:

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

where $\kappa_{\max} = Max\{w_k/\tau_k)$. $\kappa_{\max}$ denotes the maximum of the minimum execution time on each machine $m_k \quad (1 \le k \le n)$.

Computation modeling for real-time service. In real-time service, certain resources are reserved for Grid tasks so their QoS can be guaranteed. This reservation may degrade the QoS of local jobs. In queuing systems, a commonly used QoS measure is the mean waiting time, which is defined as the average waiting time of local jobs [7]. To determine whether a reservation is accepted or not, local users are naturally concerned about how much the average waiting time is affected. Let W_a denote the mean waiting time of local jobs with reservation and W_b denote the mean waiting time of local jobs without reservation. We introduce a new metric, relative slowdown (S_R) to measure the impact of resource reservation on the QoS of local jobs. It is defined as the ratio of the average waiting time with reservation and the average waiting time without reservation, W_a / W_b. The reason we don't use $W_a - W_b$ to describe the change of the average waiting time is that it reflects the absolute increased value of the waiting time. It is more appropriate to use W_a / W_b to gives the relative variation of the waiting time.

The actual waiting time depends on the underlying CPU scheduling. We have modeled the relative slowdown on two widely used CPU scheduling: first-come-first-serve (FCFS) and round-robin (RR) [8]. In FCFS, jobs are served in the order of their arrival. RR is a FCFS with a fixed quantum of time. If a job cannot finish within the time quantum it is put at the end of the waiting queue to wait for its turn again. Let κ be the reservation ratio. We have shown in an M/G/1

FCFS queuing system, the mean waiting time after reservation and the relative slowdown, are

$$W_a = \frac{1}{1-\kappa}\varphi(\lambda,\rho,\sigma,1-\kappa)\lambda^{-1}$$

$$S_R = \frac{1}{(1-\kappa)}(\frac{\varphi(\lambda,\rho,\sigma,1-\kappa)}{\varphi(\lambda,\rho,\sigma,1)}), \qquad (3)$$

respectively, where $\varphi(\lambda,\rho,\sigma,c) = \rho + \frac{\rho^2 + \lambda^2\sigma^2}{2(c-\rho)}$. $\rho = \lambda/\mu$ is machine utilization and σ is the service time standard deviation. In an M/G/1 RR queuing system, the mean waiting time after reservation and the relative slowdown, are

$$W_a = \frac{\rho}{\lambda(1-\kappa-\rho)}, \text{ and } S_R = \frac{1-\rho}{(1-\kappa-\rho)}. \qquad (4)$$

Please note that $1-\rho > \kappa > 0$ holds in the above equation..

4. Task scheduling and allocation

Most task scheduling systems, including LSF, PBS Pro, Sun Grid Engine/CODINE, and Maui Scheduler [9], are designed for dedicated and stable computing environments. The Condor system provides a matchmaking mechanism for distributed computing to allocate resources with a publish/request framework. It is aimed to match the software and hardware needs of the application, and not design to solve QoS tradeoff issues. In the PEGASUS and CHIMERA workflow management systems and in the AppLeS scheduling system [2], scheduling decisions are made based on the deterministic estimation of resource availability provided by NWS [5]. These systems assume that Grid applications and local jobs compete for resources. From the QoS point of view, these systems intend to optimize the QoS of Grid tasks under the competing QoS policy. GASA (Grid Advance Reservation API) [3], a subsystem of Globus project, provides mechanism for resource reservation so that applications can receive a certain level of service. Virtual Application Service (VAS), a deadline-bound system, makes scheduling decision based on resource reservation [10]. In these systems, a scheduling decision is made only based on the QoS requirement of the Grid application. From the QoS point of view, these systems adopt the real-time service policy without considering the impact of reservation on the QoS of local jobs.

The process of task scheduling partitions a Grid task (service) into sub-tasks (service instances) and assigns each sub-task to a selected set of resources based on the pre-developed performance model to optimize the QoS. The task partition can be achieved through either partitioning or grouping. Partitioning divides a parallel program into subtasks. Grouping joins independent tasks of a meta-task or workflow into groups and takes each group as a subtask for task allocation. The measure of scheduling is determined by the underlying QoS policy. For the best-effort service, the task scheduling is designed to optimize the performance of the Grid task. We use makespan-minimization scheduling to minimize the completion time of a Grid task. A mean-time task partition algorithm can be used to distribute the workload of a parallel program to each resource so that the difference of the mean of expected execution times of the sub-tasks is minimal. A detailed description of task partition with respect to CPU, memory, network resource heterogeneity and resource sharing is given in [2]. For the real-time service, we propose two minimization scheduling algorithms to optimize local job quality and Grid task quality respectively. For local jobs, the quantity is the relative slowdown and scheduling is done to minimize the QoS degradation of local jobs. We have proposed an equal slowdown strategy to distribute the workload of Grid tasks in order to reduce the global (maximum) relative slowdown. A failure-minimization algorithm is proposed to minimize the reservation failure probability in meeting the performance (deadline) requirement of a Grid task [8]. The scheduling concerns of the competing service are similar with that of the best-effort service. The scheduling of the constrained service and balanced service, however, become much more demanding. For the constrained service where the goal is to optimize the QoS of the Grid task while maintaining the QoS of local jobs at a certain level, we approach a scheduling algorithm by using the relative slowdown to measure a QoS degradation constraint, and using a QoS-constraint partition strategy to distribute the workload of a Grid task onto each resource as much as possible under its QoS degradation constraint. Optimal scheduling algorithms are often too costly to be used in practice. In these cases, heuristic scheduling algorithms are used to find a near optimal solution within a reasonable cost. Figure 2 gives a general form of heuristic scheduling algorithms. The basic idea is that machines are sorted first based on their potential contributions to the optimization goal. Then a local optimal can be found quickly based on this ordering. Notice that different optimization criteria could be used in finding the local optimal.

Assumption: A divisible grid task is partitioned and sub-tasks are assigned to machines appropriately for a better performance
Objective: Scheduling a grid task heuristically to reach a semi-optimal performance

----

List a set of idle machines that are lightly loaded over an observed time period.
Sort the list of idle machines in a decreasing order with their potential contributions to the optimization goal.
Generate a list of machine set by gradually adding a machine from the sorted machine list until all machines is included in a machine set
Using the binary search to find the machine set with the best value of the optimization criteria.
Assign tasks to this machine set accordingly.

Figure 2. Heuristic task scheduling algorithm

Different QoS models, task scheduling algorithms, and resource management mechanisms are needed for each resource sharing policy and application QoS requirement. A unified QoS system should be developed to provide system support of QoS delivery for different needs. While this is an extremely challenging task, we have developed a QoS testbed system based on QoS framework given in Figure 1. In the resource management, current software systems already support three resource sharing policies: real-time, competing, and best-effort. For example, Condor and Seti@home support best-effort QoS policy. They can detect resource availability by monitoring local users' activities on mouse and keyboard. In Globus and VAS [10], DSRT and classic cluster resource management system such as PBS are used to provide time-sharing/space-sharing resource reservation. Notice that, with the support of our proposed reservation-based QoS models, these resource management tools can be further extended to support the constrained policy and the balanced policy. In AppLeS project, the default process scheduling in OS on local resources is used to support the competing policy. In the PEGASUS and CHIMERA workflow management systems and in the AppLeS and GHS scheduling system [5, 9], current scheduling system can be extended by introducing the QoS-oriented task scheduling algorithms discussed in this section. The UUCS (Understanding User Comfort System) can be used for measuring performance degradation of local jobs and user comfort with resource reservation [11]. We have extended the GHS system [9] to support different QoS policies and to serve as a QoS testbed system.

5. Experimental results

We have partially implemented and tested the presented QoS policies and mechanisms. We demonstrate some testing results here in terms of performance modeling, task scheduling, and resource management for different QoS policies. For modeling, we present the results of the modeling of the best-effort service and real-time service. For task scheduling, we test the efficiency of the scheduling algorithms of the real-time service. We also carry out experiments on an actual Grid application under the real-time policy with the updated DSRT [12].

The test platforms used in our experiments are the Sunwulf cluster and the DOT Grid Testbed. 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 [9], as well as the machine usage pattern observation by researchers at Wisconsin-Madison, Maryland, Carnegie Mellon, et al [9].

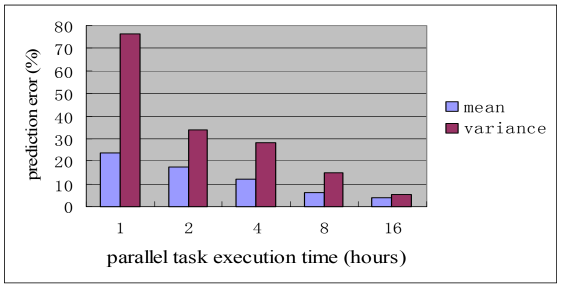

Figure 3. Mean and variance of prediction error of Cactus's execution time

Performance Modeling. The computation model for the best-effort service is designed to identify the effect of resource sharing on Grid tasks, in terms of execution time. We use the percentage prediction error to measure the prediction error. We have tested the parallel version of Cactus application on the DOT Grid Testbed. Cactus, a numerical simulation of a 3D scalar field produced by two orbiting astrophysical sources, is used as our test application. 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 ¨C which is error prone. Also the underlying DOT management system may take CPUs away from time to time. Nevertheless, Figure 3 shows the model working well even with these two factors.

A simulator of an M/G/1 queuing system is built to test the model for real-time service. The simulator is composed of a local job generator, a waiting queue, a scheduler, and a server. Both FCFS and RR queuing disciplines are supported in the simulator. The proposed model is tested with the bounded Pareto job lifetime distribution [8] as suggested by Harchol-Balter and Downey. Figure 4 plots the mean of percentage prediction error with different reservation periods from 2000 seconds to 10000 seconds when the reservation ratio is 0.2 and the utilization is 0.15. The predicted value is the calculated relative slowdown given by formula (3) and (4) derived from the proposed model and the measured value is the QoS degradation of local jobs collected from simulation results. We observe that the mean of prediction error is very small. The maximum prediction error observed in simulation is 1.15%. The mean of the prediction error tends to be smaller with the increase of the reservation period. These simulation results demonstrate that the computation model for real-time service can accurately capture the effect of reservation on the QoS degradation of local jobs [8].

We have also conducted experiments to measure the local job QoS degradation with the updated resource reservation tool, DSRT, on the DOT Grid Testbed. Two Grid applications are used in the experiments. One application is a meta-task, which consists of a set of independent indivisible computation intensive subtasks. Another application is the Cactus parallel application. In the experiment, the server and three nodes, iit01, iit02, and iit03 are used. The utilization on these machines is set as 0.2 and the reservation ratio as 0.2. FCFS queuing discipline is enforced on each resource. Table 1 gives the actual CPU part occupied by Grid tasks and the measured relative slowdown of local jobs. It shows that the measurements match the analytical result well (the calculated relative slowdown is 1.33). The small differences can be contributed to the system overhead such as synchronization and communication among processes.

Task scheduling. Task scheduling partitions and schedules a Grid task onto appropriate resources to achieve the best quality under a given policy. To test the effectiveness of the newly proposed mechanisms, a Grid environment simulator is built for the testing of scheduling under each service. The simulated Grid environment is composed of a number 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. Using the simulated Grid environment, we conducted simulations to test the efficiency of a proposed heuristic minimization scheduling algorithms for the real-time service. We simulated systems of 20, 30, and 40 machines and utilized the Weibull distribution to describe the failure probability, while our scheduling can be applied with any empirical failure probability distributions. Because local users of different machines may have different tolerances to a relative slowdown of local jobs, we differentiate the shape and scale parameters of Weibull distribution at each machine to reflect this heterogeneity by randomly generating these parameters in a certain range. We calculate the failure probabilities based on our computation model for the real-time service under five scheduling strategies: heuristic, random, fast-speed, light-load. Fast-speed and light-load scheduling means that machines are selected based on their speeds and loads respectively. For example, Fast5 represents the first 5 fastest machines are selected for task allocation. Figure 5 gives the 5th, 50th, and 95th percentiles of failure rate over 100 simulations for each scheduling policy. We can observe that not only the average reservation failure rate with heuristic scheduling is much lower than those with other scheduling algorithms, but also the variation of the failure rate with heuristic scheduling is rather smaller than others [8].

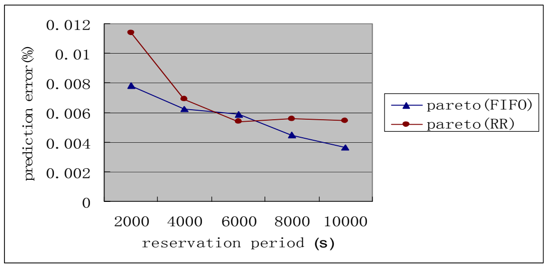

Figure 4. The mean of prediction error with different reservation periods

6. Conclusion and future work

Quality of service of Grid computing is a challenging but timely important problem. While intensive research has been conducted in QoS in recent years, most existing results are ad hoc and there is no taxonomy or a systematic understanding of the QoS under resource sharing. In this study, we conduct a thorough research to increase the fundamental understanding of QoS of Grid computing and promote system software to support QoS of Grid computing automatically. Our QoS is focused on the resource sharing of the Grid tasks and local jobs. We first introduce a QoS taxonomy based on QoS policies. The corresponding optimization issues of each QoS policies are identified next. Performance modeling and task scheduling algorithms are then presented to illustrate the solutions of selected optimization issues. Finally, some experimental results are given to confirm the feasibility and functionality of the proposed solutions. The QoS taxonomy and its associated optimization mechanisms form a framework whereas an automatic system support of QoS of Grid computing can be developed upon. In the future, we plan to further increase the fundamental understanding of quality of service of Grid computing. This may include identifying new QoS policies; in-depth understanding of existing policies; automatic policy deployment for choosing an appropriate policy under a given scenario; dynamic policy adjustment where different policies may be adopted at different time for the best fit. We plan to provide a prototype system which can automatic choose an appropriate policy, and automatic adjust QoS parameters at runtime to optimize the quality of services.

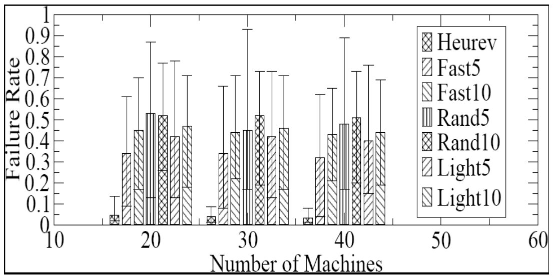

Figure 5. Failure rates with different scheduling algorithms

Reference:

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

[2] M. Wu, "System Support of Quality of Service in Shared Network Environments", dissertation, Department of Computer Science, Illinois Institute of Technology, 2006.

[3] I. Foster, A. Roy, V. Sander, "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation**",** IWQoS'00, pp. 181-188, Pittsburgh, PA, June 2000.

[4] D. Nurmi, J. Brevik, and R. Wolski, "Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments," *UCSB Computer Science Technical Report* (CS2003-28), 2003.

[5] 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.

[6] 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.

[7] D. Gross, C. M. Harris, *Fundamentals of Queuing Theory*, 3rd Edition, 1998.

[8] M. Wu, X.-H. Sun, Y. Chen, "QoS Oriented Resource Reservation in Shared Environments," *CCGrid'06*, Singapore, May 2006.

[9] M. Wu and X.-H. Sun, "Grid Harvest Service: A Performance System of Grid Computing," *Journal of Parallel and Distributed Computing,* Vol. 66, No. 10, pp. 1322-1337, 2006.

[10] K. Keahey and K. Motawi, "The Taming of the Grid: Virtual Application Services," *Technical Memorandum ANL/MCS-TM-262*, May 2003.

[11] A. Gupta, B. Lin, P. Dinda, "Measuring And Understanding User Comfort With Resource Borrowing," HPDC'2004, Honolulu, Hawaii, 2004.

[12] DSRT2.0, http://cairo.cs.uiuc.edu/software/DSRT-2/dsrt-2.html.