QoS Guided Min-Min Heuristic for Grid Task Scheduling

HE XiaoShan (何晓珊)[1], SUN XianHe (孙贤和)[1] and Gregor von Laszewski[2]

[1] *Department of Computer Science, Illinois Institute of Technology, IL, USA*

[2] *Mathematics and Computer Science Division, Argonne National Laboratory, IL, USA*

E-mail: hexiaos@iit.edu; sun@cs.iit.edu; gregor@mcs.anl.gov

Received January 7, 2003; revised March 28, 2003.

Abstract Task scheduling is an integrated component of computing. With the emergence of Grid and ubiquitous computing, new challenges appear in task scheduling based on properties such as security, quality of service, and lack of central control within distributed administrative domains. A Grid task scheduling framework must be able to deal with these issues. One of the goals of Grid task scheduling is to achieve high system throughput while matching applications with the available computing resources. This matching of resources in a non-deterministically shared heterogeneous environment leads to concerns over Quality of Service (QoS). In this paper a novel QoS guided task scheduling algorithm for Grid computing is introduced. The proposed novel algorithm is based on a general adaptive scheduling heuristics that includes QoS guidance. The algorithm is evaluated within a simulated Grid environment. The experimental results show that the new QoS guided Min-Min heuristic can lead to significant performance gain for a variety of applications. The approach is compared with others based on the quality of the prediction formulated by inaccurate information.

Keywords task scheduling, Grid computing, quality of service (QoS), non-dedicated computing

1 Introduction

Task scheduling is an integral part of parallel and distributed computing. Extensive research has been conducted in this area leading to significant theoretical[1−3] and practical results. However, with the emergence of the computational Grid, new scheduling algorithms are in demand for addressing concerns originating from the Grid infrastructure.

Traditional parallel scheduling problem schedules subtasks of an application across a parallel machine in order to reduce the turn around time while ignoring the specific shared nature of the resource. In a super computer center, the scheduling problem is enhanced by scheduling a set of applications from different users to the parallel machine while maximizing system utilization. However, in Grids the problems get amplified while scheduling across a number of such centers. Desirable goals for a Grid wide scheduling algorithm would be to increase throughput[4], maximize system utilization, and fulfill economical system and user constraints[5].

First, we discover a set of potential resources suitable for the proposed tasks. Second, we select from these resources a set that meets a predefined scheduling constraint such as minimization of the runtime. After the matching problem[6] is solved we conduct the final job execution and data transfers within the selected environment.

With the proliferation of the Grid, at least two new concepts need to be considered in a scheduling model. The first is how to handle the non-dedicated network. For non-dedicated networks, since they have their own local jobs, they cannot provide exclusive services to remote jobs. Hence, we need to address how to predict the job computation time for non-dedicated networks. The other issue is the quality of service. In a Grid environment, it is desirable to compete for the best QoS provided by and for remote resources to fulfill application constraints. The resources provide multi-level of quality of services to the applications. The scheduler in the Grid environment needs to consider application and QoS constraints to get a better match between applications and resources.

To address the challenge of non-dedicated network, we adopt the prediction model in GHS system[7] for long-term application-level performance prediction. There exist several ways to pre-

This research was supported in part by the National Science Foundation of USA under NSF Grant Nos.EIA-0224377, ANI-0123930, EIA-0130673, and by the Army Research Office under ARO Grant No.DAAD19-01-1-0432.

dict the computation time for a task/host pair. One method given in [8, 9] is to predict the current job run time using the previous performance obtained from the same or a similar host. A short-term prediction method, NWS[10], is exploited in [9] to predict the task execution time. Since the prediction is based on short-term information, for example, a 5-minute period, it may work well when the application size is small and the host environment will not change dramatically during the scheduling process. However, for large applications or dynamic running environments, an NWS-based scheduling may become unsatisfactory due to the low quality of prediction. In our scheduling design, we adopt a newly proposed long-term, application-level prediction model[11]. This prediction model was derived from a combination of rigorous mathematical analysis and intensive simulation. 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 with the guideline for performance optimization.

Our model also addressed the match of QoS request from the application and the QoS provided by the diverse resources in Grid. Research has been conducted in QoS resource management[12,13]. Most current QoS concerns, however, are at the resource management level rather than at the task/host scheduling level. In our study, we embed the QoS information into the scheduling algorithm to make a better match among different levels of QoS request/supply. Consequently, the new scheduling algorithm improves the efficiency and the utilization of a Grid system.

The organization of this paper is as follows. In Section 2, the related work is discussed. In Section 3, we introduce our scheduling algorithm. In Section 4 we present and discuss the experimental results. We conclude this study in Section 5. The implementation details of our scheduler are presented in Appendix A.

2 Related Work

Many traditional distributed algorithms[3,14] and Grid scheduling algorithms[4,15,16] have some features in common, that are performed in multiple steps to solve the problem of matching application needs with resource availability while providing quality of service.

Solving the matching problem to find the choice of the best pairs of jobs and resources is an NP-complete problem[17]. Many heuristics have been proposed to obtain the optimal match. A related traditional scheduling algorithm for scheduling problem is Dynamic Level Scheduling (DLS) algorithm[14]. In case the application to be scheduled can be structured as direct acyclic graphs for selecting the best subtask-machine pair for the next scheduling to select the best subtask-machine pair, it achieves this goal by providing a model to calculate the dynamic level as defined in [14] of the task-machine pair.

However, in a Grid environment the scheduling algorithm does not only focus on the subtasks of an application within a computational host or a virtual organization (clusters, network of workstations, etc). One of the current goals within the Grid community is to schedule a set of applications submitted in different administrative domains to the available computation power. Much research in this area has been conducted from which we point out only a few. In [1, 2], simple heuristics for dynamic matching and scheduling of a class of independent tasks onto a heterogeneous computing system have been presented. An extended suffrage heuristic was presented in [9] for scheduling parameter sweep applications based on the AppLeS[18] framework that takes the advantage of file sharing to achieve better performance under a wide variety of load conditions. The Min-Min heuristics described in [19] is becoming the benchmark of such kinds of task/host scheduling problems[2].

Currently available scheduling models include: AppLeS[9,18], Nimrod[20], and Condor[21]. The scheduling algorithm in AppLeS focuses on efficient co-location of data and experiments as well as adaptive scheduling. In addition to the prediction model adopted, our approach differs from AppLeS in that our work considers QoS in scheduling. The scheduling in Nimrod is based on different concerns from our work, which is deadlines and Grid economy model. Condor is designed for high throughput computing in a controlled local network environment. Its matchmaker scheduler[21] targets only at single processor tasks which are scheduled independently.

3 QoS-Based Grid Scheduling Model

The term quality of service (QoS) is used differently based on its context while applying it to a resource. For instance, QoS for a network may mean the desirable bandwidth for the application;

QoS for CPU may mean the requested speed, like FLOPS, or the utilization of the underlying CPU. In our study, a one-dimension QoS is considered. We represent the QoS of a network by its bandwidth.

In current Grid task scheduling, tasks with different levels of QoS requests compete for resources. While a task with no QoS request can be executed on both high QoS and low QoS resources, a task that requests a high QoS service can only be executed on a resource providing high quality of service. Thus, it is possible for low QoS tasks to occupy high QoS resources while high QoS tasks wait as low QoS resources remain idle. To overcome this shortcoming, we modify the Min-min algorithm to take the QoS matching into consideration while scheduling. Based on our selected prediction model[11], our scheduling algorithm is designed for both dedicated and non-dedicated distributed systems which are shared asynchronously by both remote and local users.

3.1 Grid and Application Model

The Grid considered in this study is composed of a number of non-dedicated hosts and each host is composed of several computational resources, which may be homogeneous or heterogeneous. To simulate the computational hosts, we profile each host with a group of local parameters, *util, arri, servstd*, to represent utilization, arrival rate, and standard deviation of service time, respectively. These parameters are the recorded performance of the computing hosts. We can read them from a file or input them. Also the QoS constraint is included in host characteristics, such as network bandwidth and latency. Different applications may require different QoS constraints.

Without the exclusive control of the local hosts, the Grid scheduler does not have control over them. The Grid scheduler must make best decisions and then submit the jobs to the hosts selected, generally as a user. Furthermore, the Grid scheduler does not have control over the set of jobs submitted to the Grid, or local jobs submitted to the computing hosts directly. This lack of ownership and control is the source of many of the problems yet to be solved in this area.

3.2 Grid Scheduling Algorithm

While there are scheduling requests from applications, the scheduler allocates the applications to the host by selecting the "best" match from the pool of applications and pool of available hosts. The selecting strategy can be based on the prediction of the computing power of the host.

3.2.1 Terminology

Throughout the paper we use terminology as follows[2,22].

The expected execution time ET_{ij} of task t_i on machine m_j is defined as the amount of time taken by m_j to execute t_i given that m_j has no load when t_i is assigned. The expected completion time CT_{ij} of task t_i on machine m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks). Let m be the total number of the machines in the HC suite. Let K be the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Let the arrival time of the task t_i be a_i, and the beginning time of t_i be b_i. From the above definitions, $CT_{ij} = b_i + ET_{ij}$. Let CT_i be CT_{ij}, where machine m_j is assigned to execute task t_i. The makespan for the complete schedule is then defined as $\max t_i \in K\ (CT_i)$. Makespan is a measure of the throughput of the heterogeneous computing system. The objective of the Grid scheduling algorithm is to minimize the makespan. Finding a makespan is NP-complete hard[17].

Existing scheduling heuristics can be divided into two categories: on-line mode and batch mode. On-line mode heuristics map a task onto a machine as soon as it arrives at the scheduler. Batch mode heuristics imply that tasks are not mapped onto the machines as they arrive; instead they are collected in a set that is examined for mapping at prescheduled times called mapping events. This independent set of tasks considered for mapping at mapping events is called a meta-task. In the on-line mode heuristics, each task is considered only once for matching and scheduling, e.g., the Minimum Completion Time (MCT) heuristic, and the Minimum Execution Time (MET) heuristic[19]. In batch mode, the scheduler considers a meta-task for matching and scheduling at each mapping event. This enables the mapping heuristics to possibly make better decisions, because the heuristics have the resource requirement information for the meta-task, and know the actual execution time of a larger number of tasks (as more tasks might complete while waiting for the mapping event).

Several simple heuristics for scheduling independent tasks are proposed in [2, 9]: Min-Min, Max-Min, Suffrage and XSuffrage. The general algorithm is shown in Fig.1[18].

```
(1)  while there are tasks to schedule
(2)      for each task $i$ to schedule
(3)          for each host $j$
(4)              Compute $CT_{i,j} = CT$ (task $i$, host $j$)
(5)          end for
(6)          Compute $metric_i = f_1(CT_{i,1}, CT_{i,2}, \ldots)$
(7)      end for
(8)          Select best metric match $(m, n) = f_2(metric_1,$
             $metric_2, \ldots)$
(9)      Commpute minimum $CT_{m,n}$
(10)     Schedule task $m$ on $n$
(11) end while
```

Fig.1. The general adaptive scheduling algorithm.

These heuristics iteratively assign tasks to processors by considering tasks not yet scheduled by computing their expected Minimum Completion Time (MCTs). For each task (line 2), this is done by tentatively scheduling it to each host (line 3), estimating the task's completion time on it (line 4). For each task, a metric function "f_1" is computed over all the hosts (line 6). Afterwards, the task/host pair with the best metric match (m, n) is selected using selection function "f_2" (line 8). We then compute the minimum completion time of this task/host pair (line 9) and assign the task m to the host n (line 10). The process is then repeated until all tasks are scheduled (line 1 and line 11).

The four heuristics, Min-Min, Max-Min, Suffrage, and Xsuffrage are defined by different definitions of "f_1" and the best metric match selection function "f_2". For example, the Min-Min, Max-Min heuristics define "f_1" as the minimum completion time, that is, for task i, they select the minimum completion time over all the hosts. However, in function "f_2", the Min-Min selects the minimum completion time over all tasks (the minimum $metric_i$), whereas the Min-Max selects the maximum completion time over all tasks (the maximum $metric_i$).

The general scheduling algorithm given in Fig.1 does not consider QoS, which affects its effectiveness in a Grid. Regardless of their computing power request, some tasks may require high network bandwidth to exchange a large amount of data among processors, whereas others can be satisfied with low network bandwidth. For example, assume that there is only one cluster (or virtual organization) that has high bandwidth in a Grid. If the scheduler assigns a task that does not require high bandwidth on the cluster, tasks requiring high bandwidth will then have to wait. Considering QoS in scheduling should lead to a better scheduling algorithm. Based on the general adaptive algorithm, a new QoS guided scheduling algorithm is proposed in Fig.2.

```
for all tasks $t_i$ in meta-task $M_v$ (in an arbitrary order)
    for all hosts $m_j$ (in a fixed arbitrary order)
                $CT_{ij} = ET_{ij} + d_j$
do until all tasks with high QoS request in $M_v$ are mapped
    for each task with high QoS in $M_v$, find a host in the QoS qualified
            host set that obtains the earliest completion time
    find the task $t_k$ with the minimum earliest completion time
    assign task $t_k$ to the host $m_l$ that gives it the earliest completion time
    delete task $t_k$ from $M_v$
    update $d_l$
    update $CT_{il}$ for all $i$
end do
do until all tasks with low QoS request in $M_v$ are mapped
    for each task in $M_v$ find the earliest completion time and the corresponding host
    find the task $t_k$ with the minimum earliest completion time
    assign task $t_k$ to the host $m_l$ that gives it the earliest completion time
    delete task $t_k$ from $M_v$
    update $d_l$
    update $CT_{il}$ for all $i$
end do
```

Fig.2. The QoS guided Min-Min heuristic.

In the algorithm, ET_{ij} denotes the expected execution time of task t_i on host m_j, defined as the amount of time taken by m_j to execute t_i given that m_j has no load when t_i is assigned. d_i denotes the next available time of host m_j, which is also the beginning time of next t_i that is to be executed on m_j. The expected completion time CT_{ij} of task t_i on host m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks).

In this QoS guided Min-Min heuristic, we consider the matching of the QoS request and service between the tasks and hosts based on the conventional Min-Min. The algorithm consists of the initialization of the completion time, and two "do" loops that schedule the high QoS tasks and low QoS tasks, respectively.

At the beginning, the QoS guided Min-Min heuristic computes the completion time of all the tasks on all the hosts (first 3 lines). Then, instead of mapping the whole meta-task to the hosts, we map the tasks with high QoS request first. In the first "do" loop, initially, for each task with high QoS request in the meta-task, the algorithm finds the earliest completion time and the host that obtains it, in all the QoS qualified host sets. Secondly, the algorithm finds the task with the minimum earliest completion time and assigns the task to the host that gives the earliest completion time to it. Thirdly, the algorithm finalizes the loop by deleting the scheduled task from the meta-task and updating d_l and CT_{il} for all i. After finishing the mapping of all the tasks with high QoS request, we map the rest of the tasks (those with low QoS request) in meta-task. In the second "do" loop, for each task with low QoS request in the meta-task, the algorithm finds the earliest completion time and the host that obtains it, in all the host sets. Secondly, the algorithm finds the task with the minimum earliest completion time, and assigns the task to the host that gives the earliest completion time to it. Thirdly, the algorithm finalizes the loop by deleting the scheduled task from the meta-task, and updating d_l and CT_{il} for all i. The second "do" loop will end till all the tasks in the meta-task are scheduled.

Table 1 gives a scenario in which the QoS guided Min-Min outperforms the Min-Min. It shows the expected execution time of four tasks on two hosts. The hosts are assumed to be idle. In this particular case, the Min-Min heuristic gives a makespan of 17 and the QoS guided Min-Min heuristic gives a makespan of 12. Figs.3 and 4 give a pictorial representation of the assignments made for the case in Table 1.

Table 1. A Sample where the QoS Guided Min-Min Outperforms the Min-Min

	m_0	m_1
t_0	5	X
t_1	2	4
t_2	7	X
t_3	3	8

X: The machine is not eligible for the task because of the low QoS it provides.

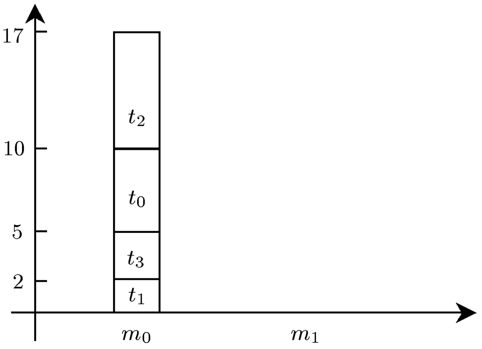

Fig.3. The Min-Min gives a makespan of 17.

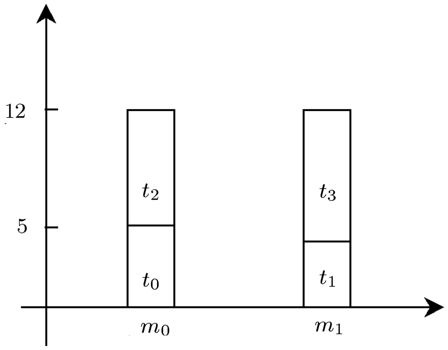

Fig.4. The QoS guided Min-Min gives a makespan of 12.

3.3 Computation Time Prediction Model

The expected task run time is a critical component of scheduling. CT_{ij} is the expected computing time of task i on host j. CT_{ij} can be computed or we assume that it can be retrieved from some database. We can calculate the value of CT_{ij} in two different ways.

- Short Term Prediction Model

During the first round of scheduling, the scheduler makes initial guess on actual task execution time for all tasks. Once tasks are completed, the scheduler uses observed execution time to improve accuracy by modifying the prediction algorithm. In AppLeS, it is shown that the average relative error of this prediction approach is about 11 percent.

- Long Term Prediction Model

A long-term performance prediction model is presented in [11] to estimate task completion time in a non-dedicated computing environment. More recently, a performance measurement and prediction system, named Grid Harvest Service (GHS)[7], has been developed based on the long-term performance model. Experimental results show that GHS provides a practical solution for long-term, application-level prediction. Its relative error is less than 10 percent.

3.4 Quality of Information

Quality of information is the impact of the performance estimation accuracy on different scheduling strategies. Different task/host selection heuristics may react differently against inaccurate estimation. Strategies with smaller vibration are more appropriate in practice. By simulating the completion time error percentage, we can investigate the tolerance of estimation accuracy of each scheduling algorithm. A good algorithm should be able to tolerate moderate prediction inaccuracy.

4 Experimental Testing

We have developed a simulated Grid environment to evaluate the newly proposed QoS guided scheduling algorithm (see Appendix A). In our experimental testing, we fixed the parameter for the hosts and used three task submission scenarios. The QoS guided Min-Min and the conventional Min-Min are compared by their makespans on the same set of tasks. At the same time, we compare the online mode scheduling and batch mode scheduling to investigate the effect of the scheduling frequency on both batch heuristics scheduling algorithms including the QoS guided Min-Min and the conventional Min-Min heuristics. In addition, we also investigate how the accuracy of the prediction algorithm affects the performance of the scheduling algorithms.

The experimental evaluation of the heuristics is performed in three parts. In the first part, the QoS guided Min-Min and the conventional Min-Min heuristics are compared with three QoS request distributions in batch mode. In the second part, the performance of the scheduling is discussed using the online mode and the batch mode with three different scheduling frequencies. And in the third part, the quality of information is discussed based on the comparison under different accuracies of prediction.

4.1 Comparison of the Batch Mode Heuristics

QoS requests have significant effect on the performance of task scheduling. Based on actual applications, three different scenarios are used in our simulation testing:

(a) Most of the tasks have particular QoS requirement. For example, 75% tasks need network bandwidth no less than 1.0Gb/s and there is no bandwidth requirement for the rest of tasks;

(b) About half of the tasks have particular QoS requirement. For example, 50% tasks need network bandwidth no less than 1.0Gb/s and there is no bandwidth requirement for the rest of tasks;

(c) Only a few of the tasks have particular QoS requirement. For example, only 25% tasks need network bandwidth no less than 1.0Gb/s and there is no bandwidth requirement for the rest of tasks.

For each of the scenarios, we compare the performance of the conventional Min-Min heuristic and that of the QoS guided Min-Min. For each scenario and each heuristic we create 100 tasks 100 times independently and get the average makespan of the 100 times. Table 2 and Fig.5 show the comparison. The data are in seconds.

Table 2. Makespan for Three Scenarios for Two Heuristics

Scenaro	Min-Min	QoS Guided Min-Min	Improvement
(a)	118.30	108.83	8.01%
(b)	152.23	134.85	11.41%
(c)	205.95	202.62	1.62%

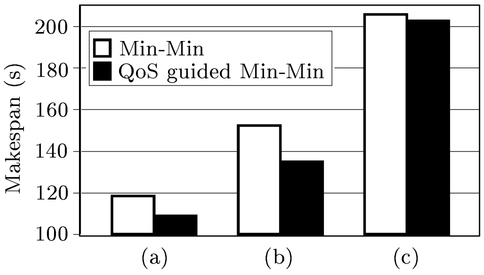

Fig.5. Makespan for three scenarios for two heuristics.

As shown in Fig.5, for all three scenarios the QoS guided Min-Min outperforms the conventional Min-Min heuristic. For our test scenarios, the makespan using QoS guided Min-Min can be as much as 11% shorter than that using the conventional Min-Min. For scenario (b) where the tasks that require high QoS and the tasks that require low QoS are evenly distributed, the performance gain reaches as high as 11.41%. For scenario (a), where the tasks that require high QoS are of higher ratio (75%), a satisfactory performance gain 8.01% is acquired. For scenario (c), where the ratio of tasks that require high QoS is only 25%, the performance gain of the QoS guided Min-Min is relatively small, i.e., 1.62% better than the conventional Min-Min.

4.2 Effect of the Scheduling Frequency in the Batch Mode Heuristics

The scheduling frequency plays an important role in the efficiency of the scheduling algorithms. If the frequency is too high, we can only get the local optimum and if the interval between scheduling events is too long, some hosts may remain idle so that the resources cannot be fully utilized.

Table 3. Makespan for Two Heuristics Based on Different Scheduling Frequency

	Min-Min	QoS guided Min-Min	Improvement
Online	150.05		
5 s	153.05	134.93	11.84%
10 s	152.23	134.85	11.41%
20 s	156.95	138.98	11.45%

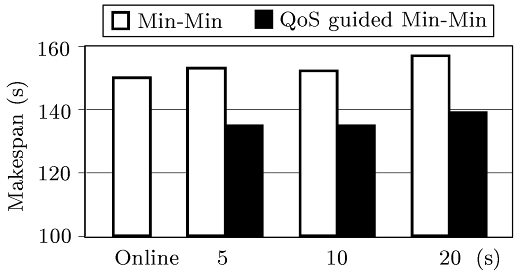

Fig.6. Makespan for two heuristics based on different scheduling frequencies.

Table 3 and Fig.6 show the comparison of the makespans of Min-Min and the QoS guided Min-Min for different scheduling frequencies including online mode. We find that for all scheduling frequencies, the makespan of the QoS guided Min-Min is 11%–12% shorter than that of the conventional Min-Min for the same set of tasks.

We can also see the impact of scheduling intervals on the makespan. We find that when the scheduling interval is 10 seconds, both the Min-Min and QoS guided Min-Min scheduling algorithms get the minimum makespan. When the scheduling frequency is high, e.g., every 5 seconds once, the makespans of both scheduling algorithms increase, though they are still better than the online mode. When the scheduling frequency is as low as once in every 20 seconds, the makespans of both Min-Min and QoS guided Min-Min increase to 156.95 and 138.98, since some of the hosts are idle for a long time during the interval between two scheduling events. In online mode, there is no difference between the new QoS guided Min-Min algorithm and the conventional Min-Min algorithm since in this mode once a task arrives it is scheduled, hence there is no priority for just one task as it is done in the new QoS guided Min-Min algorithm.

4.3 Quality of Information in the Batch Mode Heuristics

As we discussed earlier, the impact of the estimation accuracy on different scheduling heuristics is worthy of investigation. In this section, we restrict to scenario (b) to present our results. In this experiment we introduced noise to the perfectly accurate estimates of the previous experiment by adding a random percentage of the error (uniformly distributed, both positively and negatively) to the computation forecasts. The errors occur in every task within a range, e.g., 10%, 30%. We increase the error percentage by 10%, 30% and 50%, the comparison between the conventional Min-Min and the QoS guided Min-Min is shown in Table 4 and Fig.7.

Table 4. Makespan for Two Heuristics Based on Four Predication Errors

	Min-Min	QoS guided Min-Min	Improvement
0%	152.23	134.85	11.42%
10%	157.68	138.78	11.99%
30%	157.25	138.55	11.89%
50%	156.45	139.00	11.15%
Relative error	< 3.58%	< 3.08%	

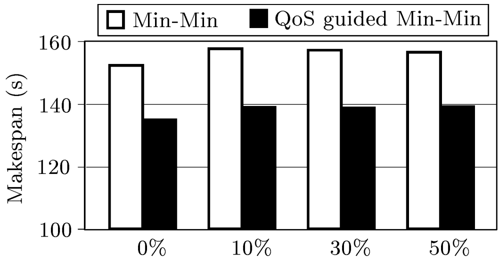

Fig.7. Makespans for two heuristics based on four prediction errors.

From Fig.7, we draw two conclusions. First, for all four prediction errors, compared with the length of their makespans, the QoS guided Min-Min outperforms the Min-Min by 11% to 12%. Second, since there is on dramatic increase (less than 3.58% and 3.08%) of the makespan as the percentage of the errors increases, one can see that both the conventional Min-Min and the QoS guided Min-Min depend on the forecast accuracy but they all can tolerate the bad quality of information to some extent. In addition, it can be seen that the QoS guided Min-Min can tolerate inaccurate estimation at least as much as the conventional Min-Min.

5 Conclusion and Future Work

To confront new challenges in task scheduling in a Grid environment, we present in this study an adaptive scheduling algorithm that considers both quality of service (QoS) and non-dedicated computing. Similar to most existing Grid task scheduling algorithms, this newly proposed scheduling algorithm is designed to achieve high throughput computing in a Grid environment. A guided QoS component is added into the conventional Min-Min heuristic to form the QoS guided Min-Min heuristic, and the most recent results in performance prediction of non-dedicated computing are used to estimate the expected execution time. A simulation system was developed to test the QoS scheduling algorithm in a simulated Grid environment. The experimental results show that the QoS guided Min-Min heuristic outperforms the traditional Min-Min heuristic in various systems and application settings. Furthermore, it also tolerates inaccurate execution estimations. Analytical and experimental results evince its real potential for Grid computing.

This study is an attempt to analyze QoS in Grid task scheduling. Many issues remain open. We have addressed only one-dimensional QoS issues. Embedding multi-dimensional QoS into task scheduling is still a topic of research. We have addressed only bandwidth constraint. We will further investigate if the concept of introducing a scheduling frequency instead of using an event-based scheduling mechanism has advantages. Finally, we intend to use this newly proposed scheduling algorithm in an actual Grid environment for practical evaluation and theoretical refinement.

References

[1] Braun T D, Siegel H J, Beck N *et al.* A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. *IEEE Workshop on Advances in Parallel and Distributed Systems*, West Lafayette, In, Oct., 1998, pp.330–335.

[2] Maheswaran M, Ali S, Siegel H J *et al.* Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. In *The 8th IEEE Heterogeneous Computing Workshop* (*HCW'99*), San Juan, Puerto Rico, Apr., 1999, pp.30–44.

[3] Al-Saqabi K, Sarwar S, Saleh K. Distributed gang scheduling in networks of heterogeneous workstations. *Computer Communications Journal,* 1997, pp.338–348.

[4] Michael Litzkow, Miron Livny, Matt Mutka. Condor — A hunter of idle workstations. In *Proc. the 8th International Conference of Distributed Computing Systems,* San Jose, CA, June, 1988, pp.104–111.

[5] Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture of a resource management and scheduling system in a global computational Grid. *High Performance Computing Asia 2000,* Beijing, China, May 14–17, 2000, pp.283–289.

[6] Papadimitriou C. Complexity Theory. Addison Wesley, Reading, MA, 1994.

[7] Xian-He Sun, Wu Ming. GHS: A performance prediction and task scheduling system for Grid computing. In *Proc. 2003 IEEE International Parallel and Distributed Processing Symposium,* Nice, France, April, 2003.

[8] Eric B Baum, Warren D Smith. Propagating distributions up directed acyclic graphs. *Neural Computation,* 1999, 11(1): 215–227.

[9] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, Francine Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Proc. the 9th Heterogeneous Computing Workshop* (*HCW'2000*), Cancun, Mexico, 2000, pp.349–363.

[10] Rich Wolski, Neil Spring, Jim Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems,* October, 1999, 15(5-6): 757–768.

[11] Gong L, Sun X H, Waston E. Performance modeling and prediction of non-dedicated network computing. *IEEE Trans. Computer*, September, 2002, 51(9): 1041–1055.

[12] Foster I, Roy A, Sander V. A quality of service architecture that combines resource reservation and application adaptation. In *Proc. the 8th Int. Workshop on Quality of Service*, Pittsburgh, PA, USA, June 5–7, 2000, pp.181–188.

[13] Diot C, Seneviratne A. Quality of service in heterogeneous distributed systems. In *Proc. the 30th Hawaii International Conference on System Sciences HICSS-30*, Hawaii, January, 1997, pp.238–247.

[14] Gilbert C Sih, Edward A Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel and Distributed Systems,* Feb., 1993, 4: 175–187.

[15] Czajkowski K, Foster I, Karonis N *et al.* A resource management architecture for metacomputing systems. In *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing,* 1998, pp.62–82.

[16] Steve Chapin, Dimitrios Katramatos, John Karpovich, Andrew Grimshaw. The legion resource management system. In *Proc. the 5th Workshop on Job Scheduling Strategies for Parallel Processing* (*JSSPP '99*), *in Conjunction with the International Parallel and Distributed Processing Symposium* (*IPDPS '99*), Germany, April 1999, pp.162–178.

[17] David Fernández-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. Software Engineering*, November, 1989, 15(11): 1427–1436.

[18] Henri Casanova, Graziano Obertelli, Francine Berman, Rich Wolski. The AppLeS parameter sweep template: User-level middleware for the grid. In *Proc. the Super Computing Conference* (*SC'2000*), Nov. 2000.

[19] Freund R F, Gherrity M, Ambrosius S *et al.* Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *The 7th IEEE Heterogeneous Computing Workshop* (*HCW '98*), Orlando, Florida, USA, Mar. 1998, pp.184–199.

[20] Buyya R, Murshed M, Abramson D. A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global Grids. In *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, Las Vegas, Nevada, USA, June 2002.

[21] Rajesh Raman, Miron Livny, Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proc. the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, July 28–31, 1998, pp.140–146.

[22] Pinedo M. Scheduling: Theory, Algorithms, and Systems. Prentice Hall, Englewood Cliffs, NJ, 1995.

HE XiaoShan is currently a Ph.D. candidate in Illinois Institute of Technology, IL, USA. She received her M.S. degree in computer science from Illinois Institute of Technology in 2002, and her B.S. degree in computer engineering from University of Electronic Science and Technology of China in 1999. Her current research interests are scheduling algorithms in grid computing and pervasive computing.

SUN XianHe received his Ph.D. degree in computer science from Michigan State University in 1990. He was a staff scientist at ICASE, NASA Langley Research Center and was an associate professor in the Computer Science Department at Louisiana State University (LSU). Currently he is a professor and the director of the Scalable Computing Software Laboratory in the Computer Science Department at Illinois Institute of Technology (IIT), and a guest faculty at the Argonne National Laboratory. Dr. Sun's research interests include parallel and distributed processing, software system, performance evaluation, and scientific computing. He has published intensively in the field and his research has been supported by DoD, DoE, NASA, NSF, and other government agencies. He is a senior member of IEEE, a member of ACM, New York Academy of Sciences, PHI KAPPA PHI, and has served and are serving as the chairman or on the program committee for a number of international conferences and workshops. He received the ONR and ASEE Certificate of Recognition award in 1999, and received the Best Paper Award from the International Conference on Parallel Processing (ICPP01) in 2001.

Gregor von Laszewski is a scientist at Argonne National Laboratory and a fellow of the Computation Institute at University of Chicago. He received a Ph.D. degree in 1996 from Syracuse University. He is an expert in Grid computing. His current research interests are in the areas of parallel, distributed, and grid computing. Specifically, he is working on topics in the area of using commodity technologies within grid services, applications, and portals.

Appendix A. The Grid Simulator

The Grid simulator was developed to test the QoS-based scheduling algorithm. The simulated test system is illustrated in Fig.8. The Grid simulator consists of a scheduler that implements different scheduling strategies in a Grid. The architecture of the Grid simulator is depicted in Fig.8.

The simulation agent generates the tasks and computational resources and sends the information to the allocation agent. The allocation agent implements two algorithms: the Min-Min and the QoS guided Min-Min. The two scheduling algorithms are tested with the same set of tasks and hosts generated by the simulation agent.

The details of the simulation agent are given in A.1. The details of the allocation agent are given in A.2. The output files are explained in A.3.

A.1 Simulation Agent

The simulation agent generates the tasks following

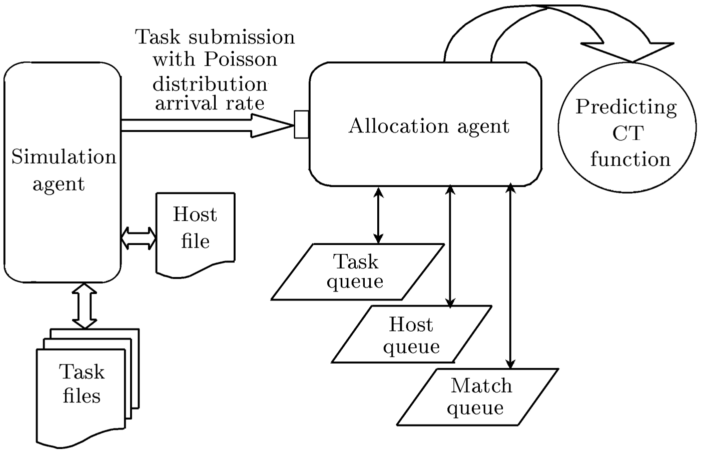

Fig.8. The scheduling model in a simulated Grid environment.

Poisson distribution. It calculates and adjusts task arrival rate λ. Each task is characterized by m subtasks and the demand of each subtask, $w_1, w_2, w_3, \ldots, w_m$. The QoS of each task is also represented in the scheduling request of the task.

To simulate the network of workstations or cluster of computing units, we profile each NOW (Network Of Workstations) or cluster with a group of parameters, *util, arri, servstd*, to represent the utilization, arrival rate, and standard deviation of service time, respectively. These parameters are the recorded average performances of the computing resources. We can read them from a file or input them manually. In our implementation, the parameters of the computing resources are generated randomly within a range. Similarly, QoS, such as network bandwidth, is generated for each computational resource.

- **Task Attributes**

100 independent tasks are created based on Poisson distribution. For each task, the attributes of the task include interarrival time, number of subtasks, QoS request, and the workloads of the subtasks. Since the tasks are created to follow the Poisson distribution, the interarrival time of tasks is generated based on exponential distribution. The number of the subtasks generated falls randomly between 1 and 20. Each subtask has a random work demand of 1,000 to 2,000. Our algorithm considers only one-dimensional QoS (network bandwidth). The QoS is generated to be 100 or 1,000 with the ratio following given distribution of the QoS request. We have three scenarios of the QoS distribution. See Subsection 3.1 for details. The run time interval is one second.

- **Host Attributes**

For each set of hosts, the attributes of the host include the number of workstations (or computing nodes), QoS provided, and the information of each workstation, which includes the utilization, arrival rate, and standard deviation of service time. The host configurations will remain the same during the simulation in our experiment. The four non-dedicated networks are settled as follows. Each network has 20 workstations and local parameters, such as *util, arri* and *servstd*, are generated randomly in given scope, e.g., *util* is from 0.0 to 1.0, arriving rate is from 0.01 to 0.15 and *servstd* is about 25. This setup makes this Grid environment non-dedicated and heterogeneous. We implement one-dimensional QoS (network bandwidth). Network_0 and network_3 have 1.0Gigabits/s bandwidth and the other two networks have 100Megabits/s bandwidth only.

A.2 Allocation Agent

The allocation agent functions in the simulated Grid environment provided by the simulation agent. The allocation agent receives the task scheduling request, schedules the tasks to the hosts, and then writes the scheduling records to the files for statistical analysis.

The allocation agent starts a listening thread that listens to the task requests. It receives the task objects and puts them into the task queue. While the task queue is not empty, the allocation starts the scheduling algorithm to find the right task/host match. To compare our QoS-guided algorithm with the Min-Min heuristic, we implement both QoS-guided heuristic and Min-Min heuristic to get the performance data.

- **QoS-Guided Heuristic**

The implementation of QoS-guided heuristic follows the algorithm shown in Fig.2. First, the allocation agent begins to compute the completion time of the task on the given host. The completion time of the task and host pair is computed by the function in [11]. The allocation agent then schedules all the tasks with high QoS request to the hosts that provide high QoS. Finally, all other tasks are scheduled on the available host set.

- **Min-Min Heuristic**

The implementation of Min-Min guided heuristic follows the algorithm shown in Fig.1. The implementation complies with the QoS rules by matching high QoS request tasks to high QoS hosts only.

A.3 Output

While scheduling, the allocation agent writes the scheduling record to files. There are three files: util.txt, allo.txt, and sim.txt. The util.txt records the utilization information of each host. The allo.txt records the scheduling information on each task, including the queuing time of each task. The sim.txt records information of the tasks that are generated. From the util.txt we can get the makespans of all the tasks.