A QoS Guided Scheduling Algorithm for Grid Computing *

Xiaoshan He[1] Xian-He Sun[1] 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

Abstract Task scheduling is an integrated component of computing. With the emergence of grid and ubiquitous computing, newer challenges have arisen in task scheduling. Unlike traditional parallel computing, grid is a shared enterprising environment where is no central control. The goal of grid task scheduling is to achieve high system throughput and to match the application needs with the available computing resources. This matching of resources in a non-deterministically shared heterogeneous environment leads to concerns on Quality of Service (QoS). In this paper we introduce a novel QoS guided task scheduling algorithm for grid computing. This new algorithm is based on the general adaptive scheduling heuristics and an added in QoS guide component. It has been tested in a simulated grid environment. The experimental results show that the new QoS guided min-min heuristic can lead to significant performance gain in various applications, and it is as tolerable as existing heuristics for inaccurate inputs of computing resources.

Key Words Task Scheduling, grid computing, Quality of Service (QoS), non-dedicated computing

1 Introduction

Task scheduling is an integrated part of parallel and distributed computing. Intensive research has been done in this area and many results have been widely accepted. However, with the emergence of the computational grid, new scheduling algorithms are in demand for addressing new concerns arising in the grid environment.

The traditional parallel scheduling problem is to schedule the subtasks of an application to the parallel machines in order to reduce the turn around time. In a grid environment, the scheduling problem is to schedule a stream of applications from different users to a set of computing resources to maximize system utilization. This scheduling involves matching of application needs with resource availability and addressing the concern of the quality of the match, that is the quality of service.

There are three main phases of scheduling on a grid [Scho02]. Phase one is resource discovery, which generates a list of potential resources. Phase two involves gathering information about those resources and choosing the best set to match the application requirements. In phase three the job is executed, which includes file staging and

* This research was supported in part by national science foundation under NSF grant EIA-0224377, ANI-0123930, EIA-0130673, and by Army Research Office under ARO grant DAAD19-01-1-0432.

cleanup. In the second phase, the choice of the best pairs of jobs and resources is a NP-complete problem [Fern89]. Many heuristics have been proposed to obtain the optimal match. A related traditional scheduling algorithm for the traditional scheduling problem is Dynamic Level Scheduling (DLS) algorithm [SiLe93]. Based on a DAG-structured application, DSL aims at selecting the best subtask-machine pair for the next scheduling. To select the best subtask-machine pair, it provides a model to calculate the dynamic level of the task-machine pair. The overall goal is to minimize the computing time of the application.

However, in a grid environment the scheduling algorithm no longer focuses on the subtasks of an application within a computational host or a virtual organization (clusters, network of workstations, etc.). The goal is to schedule all the incoming applications to the available computation power. In [MaAS99, BrSB98], some simple heuristics for dynamic matching and scheduling of a class of independent tasks onto a heterogeneous computing system have been presented. Also an extended suffrage heuristic was presented in [CaLZ00] for scheduling the parameter sweep applications which were implemented in AppLeS. It is well known that the min-min heuristics is now becoming the benchmark of such kinds of task/host scheduling problems.

There are two different goals for task scheduling: high performance computing and high throughput computing. The former aims at minimizing the execution time of each application and generally used for parallel processing, whereas the latter aims at scheduling a set of independent tasks to increase the processing capacity of the systems over a long period of time. Our approach is to develop a high throughput computing scheduling algorithm.

With the proliferation of grid, at least two new things need to be considered in a scheduling model. The first is the quality of service. In a grid environment, applications compete for the best QoS from the remote resources. The resources provide non-dedicated services to the applications. The scheduler in the grid environment needs to consider the QoS to get a better match between applications and resources. The other issue is how to handle the non-dedicated network. For non-dedicated networks, since they have their own local jobs, they cannot provide exclusive service to remote jobs. So how to predict the job computation time for non-dedicated network needs to be addressed. Research has been done on QoS and its related resource management, such as in [FoRS00, DiSe97]. The most current QoS concerns, however, are at the resource management level rather than at the task/host scheduling level. In this study, we embed the QoS information into the scheduling algorithm to improve the efficiency and the utilization of a grid system.

There are several ways to predict the computation time for a task/host pair. One method given in AppLeS [CaLZ00] is to predict the current job run time using the previous host performance. A short-term prediction method, NWS [WoSH99], is exploited in [CaLZ00]. Since the prediction is based on the short term, for example a 5 minutes period, it may work well when the application size is small and the host environment will not change

dramatically during the whole scheduling process. However, for large applications or non-stable running environments, a 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 [GoSW02].

Currently available scheduling models include: AppLeS [CaOB00, CaLZ00], Nimrod [BuMA02], and Condor [RaLS98]. 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 deadlines for a grid economy model, which has totally different concerns from our work. Condor is designed for high throughput computing in a controlled local network environment. Its scheduling concern is different from a grid environment and is not meant for stream of applications.

The organization of this paper is as follows. In section 2, our scheduling algorithm is introduced. In section 3, the experimental results are presented and discussed. We conclude this study in section 4. The implementation details of our scheduler are presented in Appendix A.

2 QoS based Grid Scheduling Model

The term quality of service (QoS) may mean different for different types of resources. For instance, QoS for 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 or without QoS request 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 prediction model [GoSW02], our scheduling algorithm is designed for both dedicated and non-dedicated distributed systems, where the distributed systems are shared asynchronously by both remote and local users.

2.1 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, representing the utilization, arrival rate, and standard deviation of service time, respectively. These parameters are

the recorded average performance of the computing hosts. We can read them from a file or input it manually. Also the QoS is included in host characteristics, such as network bandwidth and latency. Different applications may require different QoS.

The grid scheduler does not own the local hosts, therefore does not have control over them. The grid scheduler must make best effort 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.

2.2 Grid Scheduling Algorithm

While there are scheduling requests from applications, the scheduler allocates the application to the host by selecting the "best" match from the pool of applications and pool of the available hosts. The selecting strategy can be based on the prediction of the computing power of the host [GoSW02]. Before moving to the scheduling heuristics, let us review some terms and definitions [MaAS99, Pine95].

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 j is assigned to execute task i. The **makespan** for the complete schedule is then defined as $max_{ti \in K}(CT_i)$. Makespan is a measure of the throughput of the heterogeneous computing system. The objective of the grid scheduling algorithm is to minimize the makespan. It is well known that the problem of deciding on an optimal assignment of jobs to resources is NP-complete. Heuristics are developed to solve the NP-complete problem.

Existing mapping (matching and scheduling) heuristics can be divided into two categories: **on-line mode** and **batch mode**. In the on-line mode, a task is mapped onto a machine as soon as it arrives at the mapper. In the batch mode, 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 that is 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. The MCT (minimum completion time) heuristic assigns each task to the machine so that the task will have the earliest completion time [FrGA98]. The MET (minimum execution time) heuristic assigns each task to the machine that performs that task's computation in the least amount of execution time. 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 times of a larger number of tasks (as more tasks might complete while waiting for the mapping event).

Several simple heuristics for scheduling independent tasks were proposed in [MaAS99, CaLZ00]: Min-min, Max-min, Sufferage and XSufferage. These heuristics iteratively assign tasks to processors by considering tasks not yet scheduled and computing their expected Minimum Completion Time (MCTs). For each task, this is done by tentatively scheduling it to each host, estimating the task's completion time and computing the minimum completion time over all the hosts. For each task, a metric is computed using their MCTs and the task with the best metric is assigned to the host that let it achieve its MCT. The process is then repeated until all tasks have been scheduled. The general algorithm is shown in Figure 1 [CaOB00].

```
(1) while there are tasks to schedule
(2)     for all task i to schedule
(3)         for all host j
(4)             Compute CT_{i,j} = CT(task i, host j)
(5)         end for
(6)         Compute metric_i = f(CT_{i,1}, CT_{i,2}, .....)
(7)     end for
(8)     Select best metric match m
(9)     Compute minimum CT_{m,n}
(10)    Schedule task m on n
(10)end while
```

Figure 1. The General Adaptive Scheduling Algorithm

The four heuristics, MinMin, MaxMin, Sufferage, and Xsufferage, is defined by the different definitions of "*f*".

The general scheduling algorithm given in Figure 1 does not consider QoS, which affects its performance in a general grid environment. 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, let us 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 to 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 Figure 2:

(1) **for** all tasks t_i in meta-task M_v (in an arbitrary order)
(2) **for** all hosts m_j (in a fixed arbitrary order)
(3) $CT_{ij} = ET_{ij} + d_j$
(4) **do** until all tasks with high QoS request in M_v are mapped
(5) for each task with high QoS in M_v, find a host in the QoS qualified host set that obtains the earliest completion time
(6) find the task t_k with the minimum earliest completion time
(7) assign task t_k to the host m_l that gives it the earliest completion time
(8) delete task t_k from M_v
(9) update d_l
(10) update CT_{il} for all i
(11)**end do**
(12) **do** until all tasks with low QoS request in M_v are mapped
(13) for each task in M_v find the earliest completion time and the corresponding host
(14) find the task t_k with the minimum earliest completion time
(15) assign task t_k to the host m_l that gives it the earliest completion time
(16) delete task t_k from M_v
(17) update d_l
(18) update CT_{il} for all i
(19)**end do**

Figure 2. The QoS Guided Min-min heuristic

In the algorithm, ET_{ij} denotes the expected execution time of task ti on host mj, defined as the amount of time taken by mj to execute ti given that mj has no load when ti is assigned. dj denotes the next available time of host m_j, which is also the beginning time of next ti that is to be executed on mj. The expected completion time CT_{ij} of task ti on host mj is defined as the wall-clock time at which mj completes ti (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. Similar to Min-min, the QoS guided Min-min computes the completion time of all the tasks on all the hosts at the start. 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 dl and CTil for all i. After finishing the mapping all the tasks with high QoS request, we map the rest of the tasks (those with low QoS request) in meta-task. By repeating the same three steps as we described, the tasks with low QoS request are also mapped to the hosts. The QoS guided Min-min will be executed at every scheduling event.

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. Figure 3 and 4 gives 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

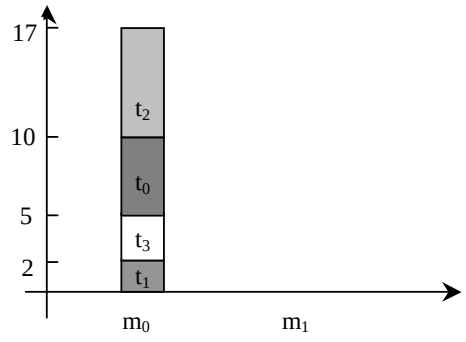

Figure 3. the Min-min gives a makespan of 17

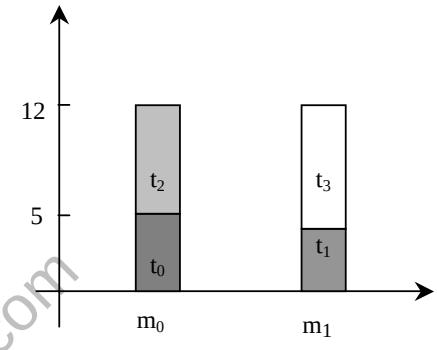

Figure 4. the QoS guided Min-min gives a makespan of 12

2.3 Computation Time Prediction Model

The expected task run time is a critical component of scheduling. CTij is the expected computing time of task i on host j. CTij can be computed or retrieved from some database. We can use two different ways to calculate the value of CTij:

- 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 times 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 [GoSW02] to estimate task completion time in a non-dedicated computing environment. More recently, a performance measurement and prediction system, named Grid Harvest Service (GHS) [SuWu02], 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.

2.4 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 on 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.

3 Experimental Testing

We have developed a simulated grid environment to evaluate the newly proposed QoS guided scheduling algorithm (see the 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 on their makespans on the same set of tasks. At the same time, we compare the online mode 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 given different accuracies of prediction.

3.1 Comparison of the batch mode heuristics

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

- Scenario (a), most of the tasks have particular QoS requirement. Such as 75% tasks need network bandwidth no less than 1.0Gigabit/s and there is no bandwidth requirement for the rest of tasks.
- Scenario (b), about half of the tasks have particular QoS requirement. Such as 50% tasks need network bandwidth no less than 1.0Gigabit/s and there is no bandwidth requirement for the rest of tasks.
- Scenario (c), only a few of the tasks have particular QoS requirement. Such as only 25% tasks need network bandwidth no less than 1.0Gigabit/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 heuristics and 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 Figure 5 shows the comparison. The data is in seconds.

Table 2. Makespan for three scenarios for two heuristics

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

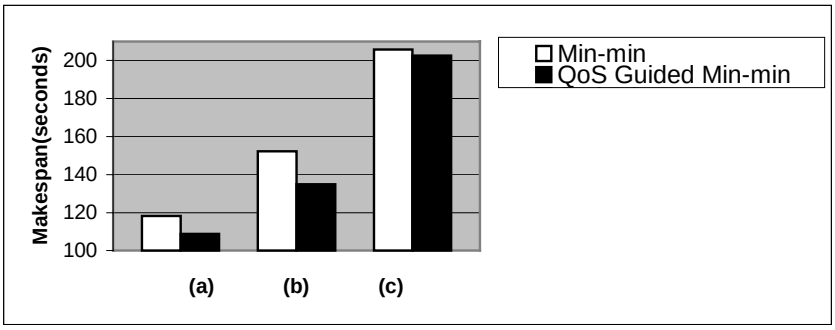

Figure 5. Makespan for three scenarios for two heuristics

As shown in Figure 5, for all three scenarios the QoS guided Min-min outperforms the conventional Min-min heuristics. The makespan using QoS guided Min-min can be as much as 11% shorter than that using the conventional Min-min. For the scenario (b) where the tasks that request high QoS and the tasks that request low QoS are evenly distributed, the performance gain reaches as high as 11.41%. For the scenario (a), where the tasks that request high QoS are in higher density (75%), a satisfactory performance gain 8.01% is acquired. For scenario (c), where the tasks that request 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.

3.2 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 secs	153.05	134.93	11.84%
10 secs	152.23	134.85	11.41%
20 secs	156.95	138.98	11.45%

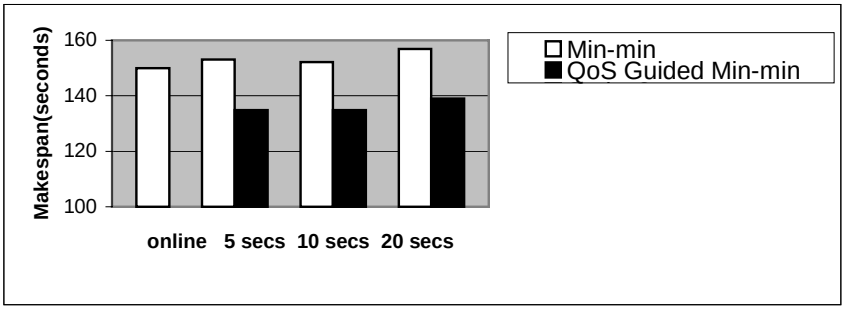

Figure 6. Makespan for two heuristics based on different scheduling frequency

Table 3 and Figure 6 show the comparison of the makespan of Min-min and the QoS guided Min-min for different scheduling frequencies including online mode. It is shown that for all scheduling frequencies, the makespan of the QoS guided Min-min is 11% – 12% shorter than that of the conventional Min-min on the same set of tasks.

We can also see the impact of scheduling intervals on the makespan. It is found 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, such as every 5 seconds once, the makespans of both scheduling algorithms increased, though 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 increased to 156.95 and 138.98, since some of the hosts are idle for long 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 has been done in the new QoS guided min-min algorithm.

3.3 Quality of Information in the batch mode heuristics

As we discussed earlier, the impact of the estimation accuracy on different scheduling heuristics should be studied. In this section, we focus on 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 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 percentage error by 10%, 30% and 50%, the comparison between the conventional Min-min and the QoS guided Min-min is show in Table 4 and Figure 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%	

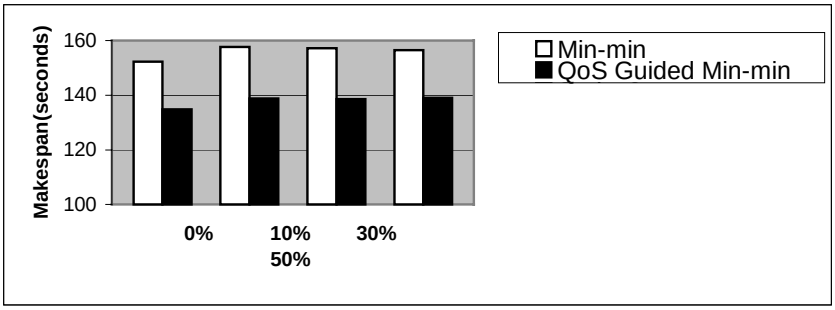

Figure 7. Makespan for two heuristics based on four predication errors

From the Figure 7, we draw two conclusions. First, for all four prediction errors, compared to the length of their makespans, the QoS guided Min-min outperforms the Min-min 11% to 12%. Second, since there is no dramatic increase (less than 3.58% and 3.08%) on 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.

4 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 is 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 system and application settings, and it also tolerates inaccurate execution estimations. Analytical and experiment results evince its real potential for grid computing.

This study is a first attempt to support 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 the concern of bandwidth of network. How to classify general QoS in a grid environment still needs more deliberation. In addition, how to choose scheduling frequency is another optimization problem that needs further investigation. Finally, we would like to adopt this newly proposed schedule algorithm in an actual grid environment for further testing and refinement.

References

[BrSB98] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, “A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems,” *IEEE Workshop on Advances in Parallel and Distributed Systems*, Oct. 1998, pp. 330-335.

[BuMA02] Buyya, R, Murshed, M., and Abramson, D. “A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids”, *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02),* Las Vegas, Nevada, USA, June 2002.

[CaLZ00] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov and Francine Berman, “Heuristics for Scheduling Parameter Sweep applications in Grid environments”, *Proceedings of the 9th Heterogeneous Computing workshop (HCW'2000),* pp349-363. 2000.

[CaOB00] Henri Casanova, Graziano Obertelli, Francine Berman and Rich Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid", *Proceedings of the Super Computing Conference (SC'2000),* Nov. 2000.

[DiSe97] *C. Diot and A. Seneviratne*. “Quality of Service in Heterogeneous Distributed Systems”. *Proceedings of the 30th Hawaii International Conference on System Sciences HICSS-30. Hawai. January 1997.*

[Fern89] David Fernández-Baca. “Allocating modules to processors in a distributed system”, *IEEE Transactions on Software Engineering*, 15(11):1427-1436, November 1989.

[FoRS00] I. Foster, A. Roy and V. Sander, A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation, *Proc. of 8th Intl Workshop on Quality of Service*, 181-188, 2000, June 5-7, Pittsburgh, PA, USA.

[FrGA98] R. F. Freund, M. Gherrity, S. Ambrosius, M. Camp-bell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, “Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet,” *7th IEEE Heterogeneous Computing Workshop (HCW ’98)*, Mar. 1998, pp. 184–199.

[GoSW02] L. Gong, X.H. Sun, and E. Waston, "Performance Modeling and Prediction of Non-Dedicated Network Computing", *IEEE Trans. on Computer,* Vol 51, No 9, September, 2002.

[MaAS99] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund, ``Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *8th*

IEEE Heterogeneous Computing Workshop (HCW '99), San Juan, Puerto Rico, Apr. 1999, pp. 30-44.

[Pine95] M. Pinedo, *Scheduling: Theory, Algorithms, and System*s, Prentice Hall, Englewood Cliffs, NJ, 1995.

[RaLS98] Rajesh Raman, Miron Livny, and Marvin Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28-31, 1998, Chicago, IL

[Scho02] Jennifer M. Schopf, "A General Architecture for Scheduling on the Grid", *special issue of JPDC on Grid Computing,* April, 2002.

[SiLe93] Gilbert C. Sih and Edward A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures", IEEE Trans. Parallel and Distributed Systems, vol.4, pp175-187, Feb.1993.

[SuWu02] Xian-He Sun, and Ming Wu, "GHS: A Performance Prediction and Task Scheduling System for Grid Computing", submitted for publication.

[WoSH99] Rich Wolski, Neil Spring, and Jim Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", Journal of Future Generation Computing Systems,Volume 15, Numbers 5-6, pp. 757-768, October, 1999.

Appendix A: The Grid Simulator

The Grid simulator was developed to test the QoS based scheduling algorithm. The simulated test system is illustrated in Figure 8. The Grid simulator consists of a simulated Grid environment and a scheduler that implements different scheduling strategies. The whole system works as Figure 8.

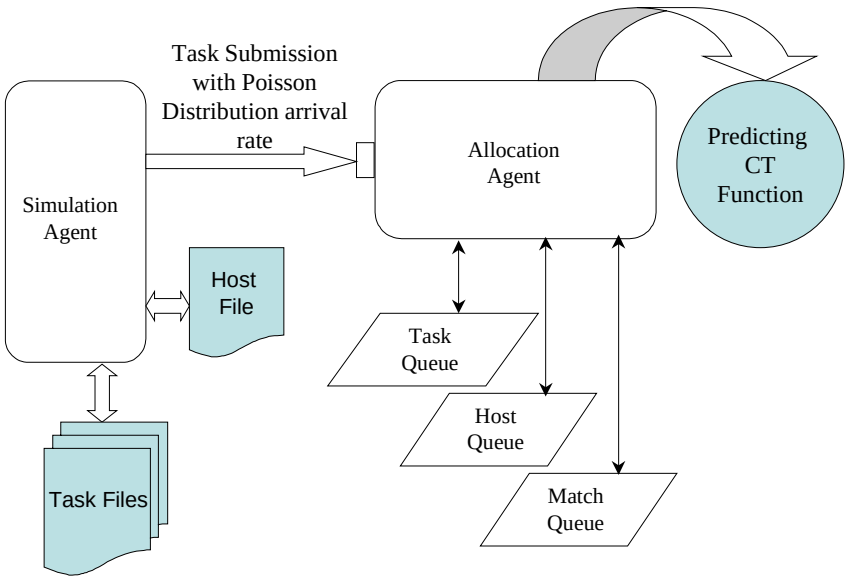

Figure 8. The scheduling model in a simulated Grid environment

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 on 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 Poisson distribution. It calculates and adjusts task arrival rate λ. Each task is characterized by m subtasks and the demand of each subtask, w1, w2, w3,…, wm. 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 or cluster with a group of parameters, util, arri, servstd, representing 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 1000 to 2000. Our algorithm considers only one dimensional QoS (network bandwidth). The QoS is generated to be 100 or 1000 with the ratio following given distribution of the QoS request. We have three scenarios of the QoS distribution. See section 3.1 for details. The run time interval is one second.

Host Attributes

For each set of hosts, the attributes of the host include 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.

All configurations about the hosts 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 the local parameters, such as util, arri and serstd, are generated randomly in given scope, such as util is from 0.0 to 1.0, arriving rate is

from 0.01 to 0.15 and serstd 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 1.0Gigabit/s bandwidth and the other two networks have 100Megabit/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 Figure 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 [GoSW02]. 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 hosts set.

Min-min heuristic

The implementation of QoS guided heuristic follows the algorithm shown in Figure 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 on the tasks that are generated. From the util.txt we can get the makespan of all the tasks.