2007－MPS－65 (14)
2007／6／25

デスクトップグリッド環境での マルチジョブスケジューリングにおける ジョブの追い越しを防ぐジョブ優先度制御

峰 久也†, 藤本 典幸†, 東 竜一†, 薄田 昌広‡, 萩原 兼一†

† 大阪大学 大学院 情報科学研究科 〒560-8531 大阪府豊中市待兼山町1-3
‡ 関西電力 株式会社 電力技術研究所 〒661-0974 兵庫県尼崎市若王寺3-11-20

内容梗概 - インターネットに接続された複数の計算機の余剰計算力を利用するデスクトップグリッドを複数のユーザで利用する場合，余剰計算力の変動を高い精度で予測することが難しいため，後から投入された計算量のより大きいジョブが，先に投入された計算量の小さいジョブも先に完了してしまうこと（ジョブの追い越し）がある．本論文では，余剰計算力が動的にどのように変化しようとも，その変動の予測の必要無しに，ジョブの追い越しが決して起こらないことを保証するスケジューリングが可能であることを示す．そしてジョブの追い越しが起こらず，かつ，余剰計算力の消費量が小さいスケジューリングアルゴリズムを提案する．提案アルゴリズムは，制約を課さないアルゴリズムとほぼ同等の性能を達成することを性能評価により示す．

Priority Control to Avoid Job Overtaking in Multiple Job Scheduling for a Desktop Grid

Hisaya Mine†, Noriyuki Fujimoto†, Ryuichi Higashi†, Masahiro Susukita‡, Kenichi Hagihara†

† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka, 560-8531, Japan

‡ Power Engineering R&D Center, The Kansai Electric Power Co., Inc.
3-11-20, Wakaouji, Amagasaki, Hyogo, 661-0974, Japan

Abstract *- Desktop grid computing is computation which uses the spare computing power of many PCs over the Internet. This spare computing power varies over time and its accurate prediction appears to be difficult. Therefore, in a multi-user desktop grid, a job submitted by some user may finish earlier than other user's jobs with smaller sizes and earlier submission times. We call such an earlier finished job an overtaking job. We deem any schedule with an overtaking job to be unfair. This paper proposes a scheduling algorithm that never results in any overtaking jobs, and does not require any prediction regarding how the spare computing power varies over time. The proposed scheduling algorithms archieve the nearly equal performance in spite of the restriction that they never result in any overtaking job.*

1 Introduction

Desktop grid computing is computation which uses the spare computing power of many PCs over the Internet. As the Internet has grown more widespread and fast, desktop grid computing has attracted attention, and many projects have obtained good results by using it.

For grid users, the time interval from job submission to job completion is one of several important factors of a job schedule. Furthermore, in a multi-user environment, fairness among users is also an important factor. In the desktop grid environment it is difficult to accurately predict variation of the spare computing power. Any scheduling algorithm without due particular consideration may cause undesirable situations for some users such as: (1) a job submitted by one user finishes earlier than a smaller, earlier submitted job from another user (we call such an earlier finished job an *overtaking job*); (2) a job submitted by some user waits for a long time before its computation is started; (3) jobs submitted by some users are apt to be assigned to lower performance PCs. In this paper, we propose a scheduling algorithm such that situation (1) never occurs. This algorithm does not require any prediction regarding how the spare computing power varies over time. That is, we solve one of the above listed unfair situations. Our algorithm can be regarded as a technique to control job priority dynamically.

2 A Grid Scheduling Model

A *task* is defined as a minimal unit for scheduling. A *job* is a set of independent tasks. In this paper, every job is independent of any other job. A job is submitted to a desktop grid by a *user*. This paper considers the situation where multiple users submit their jobs to a single desktop grid.

The *size* of a task is the number of instructions in the task. The *speed* of a processor is the number of instructions computed per unit time. A grid is heterogeneous, so processors in a grid have various speed by nature. In addition, the speed of each processor varies over time due to the load by the original users in public-resource omputing. That is, the speed of each processor is the excess computing power of the processor which is not used by the original users and is dedicated to a grid. Processor speed may be zero if the load by the original users is very heavy or the processor is powerd off.

3 Related Works

As a related job scheduling problem, one such that each job requires its own number of processors has been well studied [1-5]. For the problem, there exist several known algorithms such as FCFS (First Come First Served), LJF (Largest Job First) [6], SJF (Smallest Job First) [7], Scan [8], Backfilling [9, 10], FCFS-fill [11], LSF-RTC [9], FCFS/First-Fit [5, 11], LJF/First-Fit [5], and SJF/First-Fit[5]. However, in this paper, any job does not require the number of possessed processors. Therefore, these enormous existing algorithms do not directly compete against our algorithms.

This section gives a brief account of the first-in first-out (FIFO for short) algorithm and Round-Robin algorithm. Both algorithms are simple dynamic scheduling algorithms that do not require prediction of the spare computing power or information on jobs and tasks. In Section 5, these algorithms are compared with the proposed algorithm. For a detailed explanation of FIFO and Round-Robin, please refer to the full version [13] of this paper. In both the FIFO and Round-Robin scheduling algorithms, *overtaking jobs* can occur.

4 The Proposed Algorithms

We propose two algorithms, the *FIFO based algorithm* and the *round-robin based algorithm*. Both algorithms use job sizes and task replication, but they do not require any prediction information about available processor power.

The FIFO based algorithm selects jobs in ascending order based on submission time. Then, the algorithm allocates tasks of the selected job in an arbitrary order to processors one by one. On the other hand, the round-robin based algorithm selects jobs in a round-robin manner. Then, the algorithm allocates tasks of the selected job also in the round-robin manner to processors.

However, if the following two conditions C1 and C2 hold for a selected task T, our algorithm defers allocation of T until neither condition holds, where J is the job which includes T and where J' is a job which is smaller than J and is running at the submission time of J:

C1: J has at most B unallocated tasks.
C2: J' is still running when T is selected.

We call the above priority control technique *task deferment*. If allocation of T is deferred, then T is called a *deferred task*. The above B is called a *task deferment bound*. Notice that task deferment prevents job overtaking without any prediction of how the spare computing power varies over time.

Our algorithms allocate a task to more than one processor if the first instance of the task was allocated to a slow processor and the task is again selected to for allocation. That is, in such a case, our algorithms replicate the task. We call this technique *task replication*. Our algorithms may replicate a task more than once. The first instance is called an *original* of a task. Replicated tasks are called *replicas* of a task. The reason why our algorithms use task replication is that a replica may be completed faster than the original.

In the case of a single job with tasks of equal length, it is proved that task allocation in the round-robin order reduces the consumed spare computing power of a whole schedule [14]. Hence, our algorithms also select an allocated task in the round-robin order.

5 Simulation

First, we performed the preliminary simulation to determine whether the FIFO and Round-Robin scheduling algorithms result in overtaking jobs. In the preliminary simulation, we confirmed that overtaking jobs can occur in FIFO and Round-Robin scheduling algorithms. For a detailed explanation, please refer to the full version [13] of this paper.

Next, in order to evaluate the proposed policies we performed a simulation. In the simulation we did a total of 200 trials. In the remainder of this section, we illustrate the detail of the simulation.

5.1 Parameters

In the following section, we describe each parameter (Table 1) of our simulation.

Algorithms In the simulation, we consider the following algorithms: Round-Robin; FIFO; Round-Robin with the proposed policy; FIFO with the proposed policy.

Furthermore, for each algorithm with the proposed policy, we consider four task deferment

bounds: 1, 2, 4, and 8.

Thus, we consider ten algorithms.

PCs We assume that the grid consists of 32 PCs. We assume that PCs in offices and research institutions are used for three years. Then, based on Paranhos et al.'s performance model [15], and taking Moore's law [16] into consideration, we set the ratios for the peak-performance of the PCs at 1, 2, and 4.

Each PC has one processor, and during each time step the processor switches between low-load state and high-load state with a certain probability as shown in Table 3.

Jobs and Tasks A job consists of 128 tasks with hetero generous task lengths. All the jobs are the same construction.

32 jobs are submitted at regular intervals, we let the interval set 500, 1000, 2000, and 4000.

Performance Measures To evaluate each algorithm, we make the following measurements: computation time of a job; waiting time of a job; total time of a job; standard deviation of total time of a job.

Computation time of a job is the time from when the first task of a job starts computation to when the last task of the job finishes computation.

Waiting time of a job is the time from when a job is submitted to when the first task of the job starts computation.

Total time of a job is the time from when a job is submitted to when the last task of the job finishes computation. This is the sum of computation time and waiting time.

These three measures are to evaluate each algorithm as a scheduling algorithm.

The other measure, standard deviation of total time of a job is used to evaluate how job submission time affects total time.

5.2 Results

In the figures 1a through 4b, the vertical axis represents time, and the horizontal axis represents each algorithm. *Standard* shows the algorithm without the proposed policy, and *Bound* = n shows the algorithm with the proposed policy where the task deferment bound is n. *Interval* = *n* indicates that the jobs are submitted every n units of time.

Figure 1a and Figure 1b represent the average computation time for each algorithm. In FIFO, regardless of the interval, the average computation time varies little. This is because the computation time of a job is not affected by the presence of other jobs. On the other hand, in Round-Robin, the shorter the interval is, the longer the average computation time is. This is because a shorter interval makes computation of multiple jobs more frequent, and the number of available processors per time for a job is decreased.

Figure 2a and Figure 2b represent the average waiting time for each algorithm. In FIFO, a shorter interval tends to result in longer average waiting time. This is because tasks of a job are not assigned until the tasks of all earlier submitted jobs have been assigned. On the other hand, in Round-Robin, regardless of the interval, the average waiting time is negligible compared to the total computation time. This is because the computation time of one task is comparatively short, and the tasks of a job are assigned early without waiting for earlier submitted jobs to finish.

Table 1 Parameters of the simulation

number of jobs	8
job submission interval	500,1000,2000,4000
number of tasks	128
task size (number of instructions)	2000000~6000000 uniformly random
number of processors	32
peak performance of processor (instruction per unit time without any load)	10000,20000,40000
variation pattern of processor speed	transition between low-load state and high-load state with a certain probability
probability of transition from low-load state to high-load state	1/1000
probability of transition from high-load state to low-load state	1/100
processor speed at low-load state	95%~100%
processor speed at high-load state	0%~5%

Figure 3a and Figure 3b represent the average total time for each algorithm. In both FIFO and Round-Robin, the shorter an interval is, the longer the average total time is. In most cases, the average total time of FIFO algorithms is shorter than that of Round-Robin algorithms.

Figure 4a and Figure 4b represent the average standard deviation of total time for each algorithm. For shorter intervals, Round-Robin has a smaller standard deviation. For longer intervals, FIFO has a smaller standard deviation.

Both the FIFO and Round-Robin algorithms show no relationship of task deferment bound in regards to computation time, waiting time, total time, or standard deviation.

The proposed scheduling algorithms archieve the nearly equal performance to the standard FIFO and Round-Robin algorithms in spite of the restriction that they never result in any overtaking job.

6 Conclusions

We have proposed scheduling algorithms that never results in any *overtaking job* and we perform a simulation to evaluate the proposed algorithm.

In this simulation, we have found the algorithms with the proposed policy have nearly performance to the algorithms without the proposed policy. We have also found the algorithm based on FIFO with the proposed policy has better performance. Moreover, in terms of parallel processing of deferred tasks, we have made the simulations where task deferment bound varies, but clear tendency do not appear in computation time, waiting time, and total time.

7 References

[1] Hamscher, V., Schwiegelshohn, U., Streit, A. and Yahyapour, R.: Evaluation of Job-Scheduling Strategies for Grid Computing, IEEE/ACM International Workshop on Grid Computing (GRID2000), LNCS Vol.1971, pp.191-202, 2000

[2] Li, K.: Job Scheduling for Grid Computing on Metacomputers, IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) Workshop 4, p.180b, 2005

[3] Feitelson, D. G.: Packing Schemes for Gang Scheduling, In Job Scheduling Strategies for Parallel Processing, LNCS Vol.1162, pp.89-110, 1996

[4] Feitelson, D. G. and Jette, M. A.: Improved Unitization and Responsiveness with Gang Scheduling, In Job Scheduling Strategies for Parallel Processing, LNCS Vol.1291, pp.238-261, 1997

[5] Aida, K.: Effect of Job Size Characteristics on Job Scheduling Performance, In Job Scheduling Strategies for Parallel Processing, LNCS Vol.1911, pp.1-17, 2000

[6] Li, K.and Cheng, K. H.: Job scheduling in a Partitionable Mesh Using a Two-Dimensional Buddy System Partitioning Scheme, IEEE Transactions on Parallel and Distributed Systems, Vol.2, No.4, pp.413-422, 1991

[7] Subhlok, J., Gross, T. and Suzuoka, T.: Impact of Job Mix on Optimizations for Space Sharing Scheduler, ACM/IEEE International Conference on Supercomputing (SC'96), 1996

[8] Krueger, P., Lai, T. and Dixit-Radiya, V. A.: Job Scheduling Is More Important than Processor Allocation for Hypercube Computers. IEEE Transactions on Parallel and Distributed Systems, Vol.5, No.5 ,pp.488-497, 1994

[9] Lifka, D.: The ANL/IBM SP Scheduling System, In Job Scheduling Strategies for Parallel Processing, LNCS Vol.949, pp.295-303, 1995

[10] Skovira, J., Chan, W., Zhou, H. and Lifka, D.: The EASY - LoadLeveler API Project, In Job Scheduling Strategies for Parallel Processing, LNCS Vol.1162, pp.41-47, 1996

[11] Gibbons, R.: A Historical Application Profiler for Use by Parallel Schedulers. In Job Scheduling Strategies for Parallel Processing, LNCS Vol.1291, pp.58-77, 1997.

[12] Graham, R. L., .Bounds on Multiprocessing Timing Anomalies., SIAM Journal on Applied Mathematics, Vol.17, pp.416.429, 1969

[13] Mine, H., Fujimoto, N., Higashi, R., Susukita, M., and Hagihara, K., Priority Control to Avoid Job Overtaking in Multiple Job Scheduling for a Desktop Grid, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2007) to appear, 2007

[14] Fujimoto, N. and Hagihara, K., .Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid., International Conference on Parallel Processing (ICPP), pp.391.398, 2003

[15] Paranhos, D., and Cirne, W., and Brasileiro, F., .Trading Cycles for Information: Using replication to Schedule Bag-of-Tasks Applications on Computational Grids., International Conference on Parallel and Distributed Computing (Euro-Par), Lecture Notes in Computer Science, Vol.2790, pp.169.180, 2003

[16] Moore, G. E., .Cramming More Components onto Integrated Circuits., Electronics, Vol.38, No.8, 1965

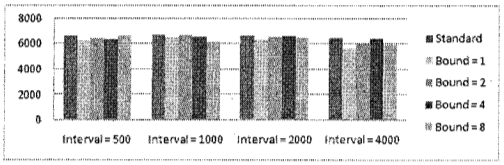

Figure 1a Computation time (FIFO)

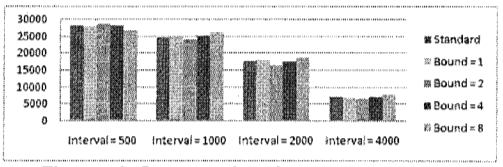

Figure 1b Computation time (Round-Robin)

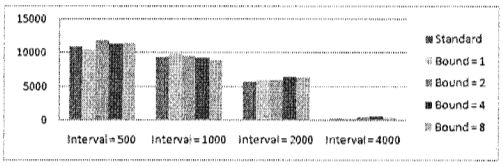

Figure 2a Waitting time (FIFO)

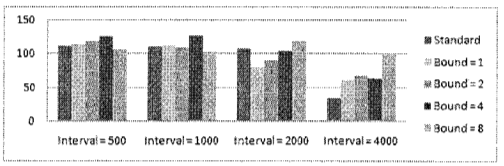

Figure 2b Waitting time (Round-Robin)

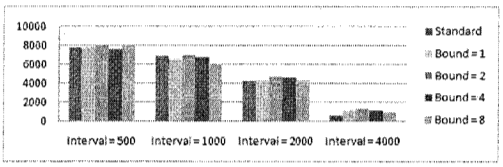

Figure 3a Total time (FIFO)

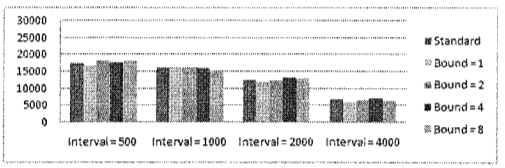

Figure 3b Total time (Round-Robin)

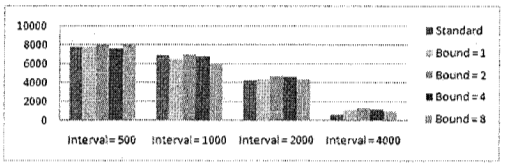

Figure 4a Standard deviation of total time (FIFO)

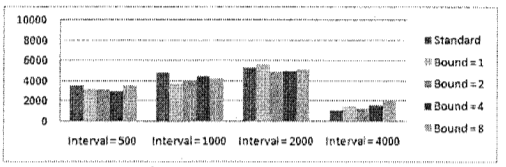

Figure 4b Standard deviation of total time (Round-Robin)