

An Integrated Processor Allocation and Job Scheduling Approach to Workload Management on Computing Grid*

Kuo-Chan Huang

*Department of Electronic Commerce
Hsing Kuo College of Management
No. 89, Yuying Street, Tainan, Taiwan
kchuang@mail.hku.edu.tw*

Hsi-Ya Chang

*National Center for High-Performance Computing
National Applied Research Laboratories
P.O. Box 19-136, Hsinchu, Taiwan
c00jhc00@nchc.org.tw*

Abstract

Processor allocation and job scheduling are two complementary techniques for improving the performance of parallel systems. This paper presents an effort in studying the issues of processor allocation and job scheduling on the emerging computing grid platform and developing an integrated approach to efficient workload management. The experimental results indicate that through careful design of processor allocation and job scheduling methods the overall system performance on computing grid can be improved more than 20 times in terms of waiting ratio.

Keywords: processor allocation, job scheduling, computing grid, workload management.

1. INTRODUCTION

In addition to cluster computing [2,3], grid computing [1] has recently become a promising trend in the high performance computing field. Many universities, research institutes, and commercial companies have been devoted to the development of related technologies and applications [4,5,6,7,8]. Among the broad range of grid computing applications, we have been focusing on the area of high throughput computing. We are concerned with improving overall system performance on a computing grid through appropriate workload management approaches. Here, workload is used to represent a mix of computing jobs

on a parallel computer during a specific period of time. It could comprise jobs of diverse characteristics.

Workload management in a multiprocessor system usually entails two steps. Job scheduling decides the job sequence for processor allocation. Processor allocation is concerned with the assignment of the required number of processors for incoming jobs. These two steps deal with different issues and aspects of workload management but both are crucial to the overall system performance.

In our previous study [11] we evaluated a few promising load sharing methods on a computing grid. The experimental results showed the *single-pool centralized queue* approach has great potential for delivering good performance on computing grid. The *single-pool centralized queue* approach regards all processors at different sites in a computing grid as a single large processor pool and thus allows a job to be allocated across site boundaries. Although allowing cross-site parallel computation can avoid the fragmentation problem, a parallel job might take much longer execution time when running across site boundaries. This is because the speed and bandwidth of inter-site network is usually much slower and less than those of intra-site network. A slower cross-site parallel job would, in turn, delay the start of execution of succeeding jobs. Therefore, the number of cross-site parallel jobs would affect the overall system performance of a computing grid in an unstraightforward way.

* This research was partly supported by the National Science Council under contracted no. NSC 94-2213-E-432-001-

This paper tries to further improve the performance of the *single-pool centralized queue* approach through dealing with the processor allocation and job scheduling issues. While allowing cross-site parallel computation, appropriate processor allocation methods are developed to reduce its frequency and job scheduling methods are designed to break the FCFS order for improving system efficiency.

2. EVALUATION OF NON-FCFS SCHEDULING POLICIES

Parallel computers are usually centrally controlled, shared by many users who might run several jobs simultaneously. Most parallel computing systems schedule parallel batch jobs based on the variable partitioning scheme [13], meaning each job receives a partition of the parallel computer with its desired number of processors. Usually the partitions of processors are allocated to the submitted jobs based on the first-come first-serve (FCFS) policy.

However, for the FCFS policy, once the number of available processors cannot meet the requirement of the first job in queue, all the jobs would be blocked until enough processors for the first job become available. This might lead to a fragmentation problem, where the available processors is less than the need of the first job but equal to or more than the needs of one or more of the jobs behind the first one. The FCFS policy does not allow jobs to run out of the order in queue. Therefore, some jobs with enough processors available are forced to wait while the processors remain idle. This fragmentation problem degrades system utilization and performance [18,19].

It is known that the best solutions for the fragmentation problem are to use dynamic partitioning [20] or gang scheduling [21]. However, these methods have practical limitations [17]. A simpler approach is to use non-FCFS policies [22], reordering the jobs in the queue. In the following, the non-FCFS policies are further divided into two categories. The first category contains backfilling based scheduling policies [14,15,16,17], and the second category refers to other non-FCFS job reordering policies.

With backfilling policies, users are asked to provide an estimate of the runtime before job submission. This enables the scheduler to predict when jobs will terminate and when the next queued jobs will be able to run. Therefore, it is possible to identify processor/time free spaces in the schedule and jobs that can fit into these spaces. Backfilling policies support two conflicting goals. The first is to move as many jobs forward as possible in order to improve utilization and

responsiveness. The second is to avoid starvation for jobs and to be able to predict when each job will run. Different versions of backfilling policies balance these goals in different ways [17].

The main problem with the backfilling policies is that they require estimates of job runtimes to be available. In this section, we propose and evaluate several non-FCFS policies that do not require users' estimates of runtimes before job submission and need much less calculation upon making scheduling decisions than backfilling policies. This section compares the effectiveness of improving system performance of the proposed non-FCFS policies against the commonly used backfilling and FCFS policies.

The following describes the evaluated non-FCFS scheduling policies in this section.

- **Backfilling.** Among the many variations of backfilling scheduling methods, the backfilling policy in this section refers to the conservative backfilling method [17]. In this policy, backfilling a specific job is done subject to checking that it does not delay any previous job in the queue. We choose the conservative backfilling method instead of other more aggressive versions such as EASY [17] because of its advantage that it allows scheduling decisions to be made upon job submission and, thus, has the capability of predicting when each job will run and giving users guaranteed response times. This is the most important feature that distinguishes the backfilling policy from other non-FCFS policies. It is clear that there is no danger of starvation for the backfilling policy as a reservation is made for each job when it is submitted.

The following are three non-FCFS policies which do not require users' estimates of runtime before job submission and need much less calculation upon making scheduling decisions than backfilling policies. Theoretically the following three non-FCFS policies run the risk of starving the large job, which requires more processors, as small jobs continue to pass it by. However, in practice this starvation problem can be avoided by allowing only a limited number of jobs to pass a job by and then starting to reserve the processors for it anyway.

- **First available.** This is the simplest non-FCFS policy proposed in this paper. In this policy each job is put into the queue according to the arrival order and at each time making a scheduling decision, the scheduler scans the queue to find a job which can run with current available processors. Therefore, the first job in the queue would not block the jobs after it from execution

even when it itself does not has enough processors for immediate execution and must wait.

- **Smallest first.** In this policy, each job is put into the queue according to its requirement of processors. Jobs with smaller requirements of processors will be put before jobs with larger requirements of processors. When two jobs have identical requirement of processors their order in the queue is determined by their arrival times. Therefore, early arrival jobs will be able to run earlier. At each time making a scheduling decision, the scheduler scans the queue to find a job which can run with current available processors. In this policy smaller jobs tend to get the requested resources earlier.
- **Largest first.** In this policy, again each job is put into the queue according to its requirement of processors. However, jobs with larger requirements of processors will be put before jobs with smaller requirements of processors. When two jobs have identical requirement of processors their order in the queue is determined by their arrival times. Therefore, early arrival jobs will be able to run earlier. At each time making a scheduling decision, the scheduler scans the queue to find a job which can run with current available processors. In this policy larger jobs tend to get the requested resources earlier.

The above three non-FCFS policies all allow the first job in queue to be passed by in order to increase the system utilization and they differ mainly in selecting which job to pass the first job by. The different job selecting preferences in these three policies can lead to different occurring patterns of free processors, which in turn will determine at any specific moment how many jobs in the queue can fit into the free processors, and, thus, affect the overall system utilization and performance.

This following describes the simulation studies we conducted to evaluate the effectiveness of the different scheduling policies described in this section for dynamic workload on a parallel computing. We used publicly downloadable workload logs on [9] as the input workload in the simulation studies. In the

following, we discuss the simulation studies with the workload log on SDSC's SP2 [23] in details.

The workload log on SDSC's SP2 contains 73496 records collected on a 128-node IBM SP2 machine at San Diego Supercomputer Center (SDSC) from May 1998 to April 2000. After excluding some problematic records based on the *completed* field [9] in the log, the simulations in this paper use 56490 job records as the input workload. The detailed workload characteristics are shown in Table 1. In the original SP2 system the jobs in this log are put into five different queues, and all these queues share the same 128 processors on the system. In the following simulations, all the 56490 jobs are assumed to be submitted to the same queue on a 384-processor computing cluster. A much larger 384-processor cluster is assumed instead of 128 processors in order to be able to simulate much heavier workloads. To evaluate the effectiveness of the scheduling policies from light to heavy workloads, our studies conduct simulations for workloads with different load factors: 1.0, 1.5, 2.0, 2.5, 3.0, 4.0. Load factor 1 represents the original workload described in Table 1 and the workloads with the other load factor values are formed by multiplying the runtime of each job by the corresponding load factor value.

We evaluated the scheduling policies in terms of average waiting time and waiting ratio. We defined *waiting ratio* to be the ratio of waiting time to execution time [12]. Smaller waiting ratios usually imply higher user's satisfaction. The concept of waiting ratio is similar to the notion of bounded slowdown in [22]. However, in our definition of waiting ratio, we exclude the runtime in the nominator. Table 2 and Table 3 show the performance results for different scheduling policies under workloads with different load factor values, respectively. The results indicate that all non-FCFS policies outperform the FCFS policy and in general the *smallest first* policy is superior. In terms of waiting time, both *smallest first* and *first available* policies perform better than the *backfilling* policy. As regarding the waiting ratio, only the *smallest first* policy outperforms the *backfilling* policy for all workloads.

Table 1. Characteristics of the workload log on SDSC's SP2

	Number of jobs	Maximum execution time (sec.)	Average execution time (sec.)	Maximum number of processors per job	Average number of processors per job
Queue 1	4053	21922	267.13	8	3
Queue 2	6795	64411	6746.27	128	16
Queue 3	26067	118561	5657.81	128	12
Queue 4	19398	64817	5935.92	128	6
Queue 5	177	42262	462.46	50	4
Total	56490				

Table 2. Average waiting time for different scheduling policies

	FCFS	First Available	Smallest First	Largest First	Backfilling
Load Factor=1.0	102.71	49.17	47.40	51.86	50.73
1.5	335.20	142.49	134.14	150.79	157.63
2.0	956.25	370.18	343.39	417.00	396.85
2.5	2815.84	938.98	833.67	1075.84	1082.79
3.0	8328.50	2280.82	1909.66	3059.78	2863.94
4.0	350945.34	51191.40	49653.18	95794.07	65481.10

Table 3. Average waiting ratio for different scheduling policies

	FCFS	First Available	Smallest First	Largest First	Backfilling
Load Factor=1.0	0.77	0.17	0.15	0.21	0.16
1.5	1.89	0.51	0.40	0.53	0.53
2.0	4.08	0.86	0.79	1.15	0.85
2.5	10.38	1.68	1.29	2.03	1.75
3.0	26.92	3.15	2.19	5.27	3.22
4.0	840.93	62.63	21.99	153.65	50.72

3. PROCESSOR ALLOCATION METHODS FOR REDUCING CROSS-SITE PARALLEL COMPUTATION

In our previous study [11], in general, *the single-pool centralized queue* method would first try to schedule an entire parallel job onto a single cluster at a certain site. Only when no single cluster can accommodate such job, the method then arranges that parallel job to run across site boundaries. However, it is not uncommon that at the same time there are more than one cluster which are all be able to accommodate an entire parallel job. In that situation, although different choices of clusters make no difference to the job itself, different choices can lead to different probabilities for the succeeding jobs to become cross-site parallel computation.

The processor allocation problem for reducing cross-site parallel jobs can be further divided into two parts. The first part is how to choose an appropriate cluster for a parallel job among several clusters which all can fulfill the resource requirement of that job. The second part is to select a set of clusters among all the clusters in the computing grid for a cross-site parallel job. In the following we begin with dealing with the first part. The first part of the processor allocation problem is somehow similar to the dynamic storage allocation problem, which is how to satisfy a request of size n from a list of free holes in the storage system [10]. The set of holes is searched to determine which hole is best to allocate. *First-fit*, *best-fit*, and *worst-fit* are the most common strategies used to select a free hole from the set of available holes [10]. In the following simulations, we

evaluate five different allocation methods. In addition to the three common strategies, we evaluate two more methods: *median-fit* and *random-fit*. The following briefly describes how these five allocation methods search for a site in a computing grid to allocate.

- **First-fit.** Allocate the first site found in the searching process that has enough available processors for the waiting job. The system can stop searching as soon as it finds a site that has enough available processors, and therefore this method is generally faster.
- **Best-fit.** Allocate the site with the least available processors that is enough for the waiting job [24,25]. The system must search each site in the computing grid. This method produces the least leftover processors for the selected site.
- **Worst-fit.** Allocate the site with the most available processors that can fulfill the requirement of the waiting job. Again, the system must search each site in the computing grid. This method produces the most leftover processors for the selected site which may be more likely to accommodate the next waiting job.
- **Median-fit.** Among all the sites that have enough processors for the waiting job, allocate the site with the median number of available processors. This method also needs to search each site in the computing grid to find the median site.

- **Random-fit.** Randomly allocate a site among all the sites that have enough available processors for the waiting job. The system first search the computing grid to find all those sites that have enough available processors, and then pick up one randomly.

In the following simulations, the workload log and performance metrics used are the same as described in section 2. In the SDSC's SP2 system the jobs in this log are put into five different queues and all these queues share the same 128 processors on the system. In the following simulations this workload log will be used as the workload on the computing grid, where the grid consists of five different sites whose workloads correspond to the jobs submitted to the five queues respectively. To focus on the effects of cross-site parallel computation, in the simulations we assume the computing clusters at these five sites have the same type of processors and interconnection networks with the same speed. However, clusters at different sites may have different number of processors. Table 4 shows the configuration of the computing grid for the simulations.

Table 4. Configuration of the computing grid

	total	site 1	site 2	site 3	site 4	site 5

Table 5. Average waiting time for different slowdown ratios (sec.)

Slowdown ratio	First Fit	Median Fit	Random Fit	Best Fit	Worst Fit
1	50.36	50.36	50.36	50.36	50.36
2	61.71	65.08	80.10	60.48	93.93
4	117.57	166.13	92.48	64.05	530.58
5	200.71	959.88	133.83	99.44	2219.90

Table 6. Average waiting ratio for different slowdown ratios

Slowdown ratio	First Fit	Median Fit	Random Fit	Best Fit	Worst Fit
1	0.37	0.37	0.37	0.37	0.37
2	0.47	0.50	0.65	0.47	0.76
4	0.90	1.28	0.86	0.51	5.06
5	1.88	10.11	1.14	0.75	23.11

The followings deal with the second part of the processor allocation problem, how to allocate cross-site parallel jobs. Since the results in Tables 5 and 6 indicate that the *best-fit* method is the best one, we decide to adopt the *best-fit* policy when allocating cross-site parallel jobs. The *best-fit* policy for allocating cross-site parallel job is to select a set of sites in the computing grid such that the goal of the least leftover processors for the selected sites is achieved. This becomes a combinatorial optimization problem. Therefore, to find a set of sites that lead to the least leftover processors is much more difficult and time-consuming than finding a site with the

Number of processors	442	8	128	128	128	50
----------------------	-----	---	-----	-----	-----	----

Tables 5 and 6 show the average waiting time and waiting ratio of the five processor allocation methods under different slowdown ratios. The slowdown ratio represents the likely slowdown due to slower inter-site communication when executing a parallel job across site boundaries. It is defined as $\frac{ExecutionTimeAcrossSiteBoundaries}{ExecutionTimeWithinSingleSite}$ [11].

The results for both performance metrics reflect the same trend. When the slowdown ratio is equal to one, all sites in the computing grid are like a big cluster. Therefore, the five methods show no performance difference. In general, the *best-fit* method outperforms the other four methods. When the slowdown ratio is less than or equal to two, the performance order of these five methods from high to low is *best-fit* > *first-fit* > *median-fit* > *random-fit* > *worst-fit*. However, when the slowdown ratio is larger than or equal to four, the *random-fit* method beats the *first-fit* and the *median-fit* methods, and the performance order changes to *best-fit* > *random-fit* > *first-fit* > *median-fit* > *worst-fit*.

least leftover processors in the first part of the processor allocation problem.

In this paper we adopt heuristic methods for the site selection issue to avoid the time-consuming computation for solving the combinatorial optimization problem and to make the site selection process more practical to be applied to the real world. The followings evaluate three different heuristic methods.

- **Fixed order.** The system maintains a list of all the sites in the computing grid. The order of this list is fixed. Each time the system picks up a site from the list according to the fixed order until the total

number of available processors on all selected sites is larger than or equal to the requirement of the waiting parallel job. This heuristic method is used for dealing with cross-site parallel jobs in Tables 5 and 6.

- **Larger first.** The system first sorts the sites in the computing grid into decreasing order according to the number of available processors. Then, the system repeatedly picks up a site according to the sorted order until the total number of available processors on all selected sites is larger than or equal to the requirement of the waiting parallel job [24].

- **Smaller first.** In contrast to *larger-first*, the system first sorts the sites in the computing grid into increasing order according to the number of available processors. Then, the system repeatedly picks up a site according to the sorted order until the total number of available processors on all selected sites is larger than or equal to the requirement of the waiting parallel job.

The following simulations evaluate these three heuristics. Tables 7 and 8 show that in general the *larger-first* heuristic outperforms the other two heuristics.

Table 7. Waiting time for different heuristic methods (sec.)

Slowdown ratio	Best Fit with Fixed Order	Best Fit with Smaller First	Best Fit with Larger First
1	50.36	50.36	50.36
2	60.48	60.13	60.23
4	64.05	64.07	63.71
5	99.44	176.57	66.12

Table 8. Waiting ratio for different heuristic methods

Slowdown ratio	Best Fit with Fixed Order	Best Fit with Smaller First	Best Fit with Larger First
1	0.37	0.37	0.37
2	0.47	0.47	0.47
4	0.51	0.51	0.51
5	0.75	1.61	0.53

4. AN INTEGRATED APPROACH TO WORKLOAD MANAGEMENT ON COMPUTING GRID

In this section we propose an integrated workload management approach considering both of the processor allocation and job scheduling issues on computing grid. Based on the studies described in the previous sections, the integrated approach adopts the *smallest first* policy

for job scheduling and uses the *best-fit with larger first* policy for processor allocation. Table 9 presents the performance comparison of the integrated approach and the original *single-pool centralized queue* method in [11]. The results indicate that the integrated approach greatly improves the performance of the *single-pool centralized queue* method and the improvement increases as the slowdown ratio gets larger. The integrated approach achieves more than 20 times performance improvement in terms of average waiting ratio when the slowdown ratio equals 5.

Table 9. Performance comparison of the integrated approach and single-pool centralized queue method

Slowdown ratio	Average waiting time (sec.)		Average waiting ratio	
	Single-pool centralized queue	The integrated approach	Single-pool centralized queue	The integrated approach
5	200.71	28.54	1.88	0.08
4	117.57	23.68	0.90	0.07
2	61.71	23.30	0.47	0.07
no slowdown	50.36	22.18	0.37	0.06

5. CONCLUSIONS

This paper studies two important issues, processor allocation and job scheduling, of workload management

on computing grid. We evaluated and analyzed several processor allocation and job scheduling methods with simulation studies. Based on the studies, we propose an integrated workload management approach, considering both processor allocation and job scheduling, to improve

the performance of the previously proposed *single-pool centralized queue* method for efficient load sharing on computing grid. While allowing cross-site parallel computation, appropriate processor allocation methods are developed to reduce its frequency and job scheduling methods are designed to break the FCFS order for improving system efficiency. The *single-pool centralized queue* method, as shown in our previous studies [11], can reduce the average waiting time to less than one tenth of that of a system without applying grid computing technology when the slowdown ratio is smaller than 5. The integrated approach, as shown in the simulation studies in this paper, can further achieve more than 20 times performance improvement in terms of average waiting ratio when the slowdown ratio equals 5. Moreover, the improvement increases as the slowdown ratio gets larger.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the National Center for High-Performance Computing in providing resources under the national project, "Taiwan Knowledge Innovation National Grid".

REFERENCES

- [1] Foster, I., Kesselman, C., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Inc., 1999.
- [2] Buyya, R., *High Performance Cluster Computing: Architectures and Systems, Volume 1*, Prentice Hall PTR, 1999.
- [3] Buyya, R., *High Performance Cluster Computing: Programming and Applications, Volume 2*, Prentice Hall PTR, 1999.
- [4] The globus project, <http://www.globus.org/>
- [5] Global Grid Forum, <http://www.gridforum.org>
- [6] Platform, <http://www.platform.com/>
- [7] IBM Grid Computing, <http://www-106.ibm.com/developerworks/grid/>
- [8] Sun ONE Grid Engine, <http://www.sun.com/software/gridware/sge.html>
- [9] Parallel Workloads Archive, <http://www.cs.huji.ac.il/labs/parallel/workload/>
- [10] Silberschatz, A., Peterson, J., Galvin, P., *Operating System Concepts*, Addison-Wesley Publishing Company, 1991, pp. 106-113.
- [11] Huang, K. C., and Chang, H. Y., "Performance Evaluation of Load Sharing Policies on Computing Grid", *The 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05)*, Las Vegas, USA, June 27-30, 2005, pp. 217-223.
- [12] Huang, K. C., "Scheduling Jobs to Keep Reasonable Waiting Time on Parallel Computers", *Proceedings of the First Workshop on Grid Technologies and Applications*, December 7-8, 2004, Hsinchu, Taiwan.
- [13] Feitelson, D. G., "A Survey of Scheduling in Multiprogrammed Parallel Systems", Research Report RC 19790 (87657), IBM T. J. Watson Research Center, Oct. 1994.
- [14] Feitelson, D. G. and Weil, A. M., "Utilization and Predictability in Scheduling the IBM SP2 with Backfilling", *Proc. 12th Int'l Parallel Processing Symp.*, pp. 542-546, Apr. 1998.
- [15] Gibbons, R., "A Historical Application Profiler for Use by Parallel Schedulers", *Job Scheduling Strategies for Parallel Processing*, pp. 58-77, Springer-Verlag, 1997.
- [16] Lifka, D., "The ANL/IBM SP Scheduling System", *Job Scheduling Strategies for Parallel Processing*, pp. 295-303, Springer-Verlag, 1995.
- [17] Mu'alem, A. W. and Feitelson, D. G., "Utilization, Predictability, Workloads, and User Runtime Estimate in Scheduling the IBM SP2 with Backfilling", *IEEE Transactions on Parallel and Distributed Systems*, Vol.
- [18] Krueger, P., Lai, T. H., Radiya, V. A., "Job Scheduling is More Important Than Processor Allocation for Hypercube Computers", *IEEE Transactions on Parallel and Distributed Systems*, May 1994, pp. 488-497.
- [19] Jones, J. P. and Nitzberg, B., "Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization", *Job Scheduling Strategies for Parallel Processing*, pp. 1-16, Springer-Verlag, 1999.
- [20] McCann, C., Vaswani, R. and Zahorjan, J., "A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors", *ACM Trans. Computer Systems*, Vol. 11, No. 2, pp. 146-178, May 1993.
- [21] Feitelson, D. G. and Jette, M. A., "Improved Utilization and Responsiveness with Gang Scheduling", *Job Scheduling Strategies for Parallel Processing*, pp. 238-261, Springer-Verlag, 1997.
- [22] Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P., "Theory and Practice in Parallel Job Scheduling", *Job Scheduling Strategies for Parallel Processing*, pp. 1-34, Springer-Verlag, 1997.
- [23] The JOBLOG data is Copyright 2000 The Regents of the University of California All Rights Reserved.
- [24] Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., "On Advantages of Grid Computing for Parallel Job Scheduling", *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin, May 2002.
- [25] Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R., "Evaluation of Job Scheduling Strategies for Grid Computing", *Lecture Notes in Computer Science LNCS 1971*, Springer Verlag, 2000.