2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks

Improving Processor Allocation in Heterogeneous Computing Grid through Considering Both Speed Heterogeneity and Resource Fragmentation

Po-Chi Shih
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
e-mail:
shedoh@sslab.cs.nthu.edu.tw

Kuo-Chan Huang
Department of Computer and Information Science
National Taichung University
Taichung, Taiwan
e-mail: kchuang@mail.ntcu.edu.tw

Yeh-Ching Chung
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
e-mail: ychung@cs.nthu.edu.tw

***Abstract*—In a heterogeneous grid environment, there are two major factors which would severely affect overall system performance: speed heterogeneity and resource fragmentation. Moreover, the relative effect of these two factors changes with different workload and resource conditions. Processor allocation methods have to deal with this issue. However, most existing allocation methods focus on one of these two factors. This paper first analyzes the relative strength of different existing methods. Based on the analysis, we propose an intelligent processor allocation method which considers both the speed heterogeneity and resource fragmentation effects. Extensive simulation studies have been conducted to show that the proposed method can effectively deliver better performance under most resource and workload conditions.**

Keywords-grid; speed heterogeneity; resource fragmentation; processor allocation

I. INTRODUCTION

Both job scheduling [12,15] and processor allocation [2,8] received a lot of research attention on earlier hypercube-based parallel computers. Job scheduling determines the sequence of starting execution for the jobs waiting in the queue. On the other hand, processor allocation chooses an appropriate portion of the free processors in a system for allocating the first job in the queue. On a hypercube computer, allocating a job to different sub-cubes, although having little or no impact on that single job's performance, might lead to diverse overall system performance. This is because different allocation decisions lead to different distributions of leftover processors and, in turn, different probabilities of successful allocation of subsequent jobs. The different probabilities of successful allocation usually comes from situations called *resource fragmentation* where no single sub-cube can accommodate a job while the total number of free processors in the system is equal to or larger than the requirement of the job. Therefore, good processor allocation methods, which can alleviate resource fragmentation, were helpful to system performance then.

Later, when switch-based parallel computers and cluster-based computing systems being widely used, job scheduling became a more important issue than processor allocation. This stemmed from the fact that on such systems allocation can be made with any portion of the system and with any number of processors, in contrast to the power-of-two restriction on earlier hypercube computers. Therefore, the resource fragmentation problem was eliminated and processor allocation seemed straightforward. Many research efforts [6,7,10,13,14] have been spent on the job scheduling issue on such switch-based parallel computers or cluster-based computing systems.

However, as *grid* [1,3] becomes a promising computing platform, the resource fragmentation problem is coming back again and processor allocation needs to deal with it. A computing grid usually consists of several parallel or cluster computers located at different sites. Communications between processors within the same site are usually achieved through high-speed networking devices, while messages passed across different sites have to go through a much slower wide-area network or Internet. A job allocated to a pool of processors within the same site can usually run faster than if it is assigned to processors across different sites. Therefore, the system tends to allocate a job within a single site to achieve high performance. This allocation policy could lead to resource fragmentation when no single site can accommodate a parallel job while the total number of free processors in all sites is enough for the job's execution. Processor allocation methods can be carefully designed to reduce the probability of resource fragmentation and thus increase system performance.

The *best-fit* processor allocation method has been demonstrated to be the best choice in a homogeneous grid in previous works [4,5]. For the *best-fit* method a particular site is chosen for a job on which the job will leave the least number of free processors if it is allocated to that site. Although the *best-fit* method can effectively alleviate the resource fragmentation problem, it cannot achieve good performance in a heterogeneous grid as shown in [9]. This is because in a heterogeneous grid resource fragmentation is not the sole factor that affects the overall system performance. Speed-heterogeneity is another important issue to consider. This paper tries to improve processor allocation methods in heterogeneous grid environments by considering both speed heterogeneity and resource fragmentation. A new processor allocation method was developed and extensive experiments under different workload conditions were conducted to evaluate the new method, together with other processor allocation methods for grid environments.

It is believed that no single processor allocation method can always perform the best under all possible workload conditions. However, carful and extensive analysis of the

978-0-7695-3908-9/09 $26.00 © 2009 IEEE
DOI 10.1109/I-SPAN.2009.14

performances of different methods under various workload conditions could lead to better understanding of the root causes of the performance difference of the methods. The understanding could in turn help develop more effective processor allocation methods.

The remainder of this paper is organized as follows. In section II, we analyze the potential strength of existing allocation methods in the first part and present the proposed *intelligent* allocation method in the second part. Section III present and discuss the result of our experiment. Conclusion of this paper is given in section IV.

II. PROCESSOR ALLOCATION METHODS IN HETEROGENEOUS GRID

In this section we begin by analyzing the pros and cons of existing processor allocation methods. The analysis then guides us to the development of a more effective processor allocation method.

A. Analysis of Existing Processor Allocation Methods

The *best-fit* method [4,5] allocates a job to the site which can yield the smallest resource fragmentation. This scheme works fine in a homogeneous grid. However, in a heterogeneous grid with computing speed differences among participating sites, the *best-fit* method may not perform well since it does not consider the speed heterogeneity [9]. In such an environment another processor allocation method called *fastest-first* has been proposed [9]. The *fastest-first* method focuses on speed heterogeneity in a heterogeneous grid and allocates a job to the fastest one among all the sites which can accommodate the job. Since *fastest-first* does not consider the difference between the amount of required processors and a site's free capacity, it may result in larger fragmentation than *best-fi*t.

Besides, the relative performance of these two methods would largely depend on several factors such as computing speed heterogeneity, system loading, workload condition, and so on. Speed heterogeneity is measured by the variance of computing speeds of all participating sites in a grid. System loading can be simply observed and represented by the average length of the job waiting queue. Workload condition includes many attributes such as job arrival process, probability distribution of the numbers of required processors, execution time distribution, etc. Some of these parameters can be seen as random variables that dynamically change with time (e.g., system loading and workload condition). It is hard to have any allocation method that can surpass all other methods in all workload conditions. To this end, we focus on identifying the potential strength of each allocation method under different conditions and trying to combine all the advantages to form a new allocation method. We expect this new allocation method can achieve better performance in all workload conditions.

Table I shows the relative strength analysis of *best-fit* and *fastest-first* under different levels of speed heterogeneity and system loading. Since *best-fit* does not consider speed difference among participating sites, it is more suitable to be used when speed heterogeneity is low. Additionally, *best-fit* were shown to yield less resource fragmentation and lead to higher resource utilization than *first-fit* method, which inspects the participating sites in a fixed order and allocates a job to the first site found to be able to accommodate the job [5]. Since *fastest-first* can be viewed as another form of *first-fit* if the sites in a grid are arranged in the descending order of computing speed, *best-fit* can be expected to outperform *fastest-first* in reducing resource fragmentation and raising resource utilization. When system loading is high, resource utilization rate is crucial to the overall system performance. Therefore, *best-fit* has higher potential to perform better than *fastest-first* when system loading is high. It is then clear that in the case of low speed heterogeneity and high system loading, *best-fit* is a better choice. On the contrary, when resource heterogeneity is high and system loading is low (one can image the extreme case when the waiting queue length is 0), computing speed of each job has higher influence on the overall system performance than the resource fragmentation effect. Therefore, *fastest-first* can potentially perform better than *best-fit* in this case.

We use a parentheses pair to represent speed heterogeneity and system loading. For example, (low, high) represents a situation that heterogeneity is low and loading is high. In table I, we only list the potentially best allocation method in the (low, high) and (high, low) situations. For the cases (low, low) and (high, high), it is hard to tell which one is better. Based on the above analysis, we begin to develop a new approach named *intelligent* allocation by considering both speed heterogeneity and resource fragmentation effects in the following section.

TABLE I. RELATIVE STRENGTH ANALYSIS OF DIFFERENT ALLOCATION METHODS

	System Loading(High)	System Loading(Low)
Speed Heterogeneity (high)	undistinguishable	*fastest-first*
Speed Heterogeneity (low)	*best-fit*	undistinguishable

B. Intelligent Allocation method

This section presents the proposed *intelligent* allocation method. The main idea behind the method is to dynamically switch the allocation decision between *best-fit* and *fastest-first* according to some measurable criteria. To clarify the following presentation, we first define several terms as follows.

- *Waiting Queue* (*WQ*) – the queue which contains all jobs waiting for available resources in its arriving order.
- *Size of WQ* ($Size_{WQ}$) – total number of jobs in the waiting queue.
- *Required number of processors* (RNP_i) – the required number of processors of job *i*.
- *Computing Speed* (CS_j) – the computing speed of site *j*.
- *Number of free Processors* (NP_j) – the number of free processors in site *j*.
- *Site selected by best-fit* ($S_{bf}(i)$) – the site allocated for computing job *i* by the *best-fit* method

- *Site selected by fastest-first* ($S_{ff}(i)$) – the site allocated for computing job i by the *fastest-first* method.
- *The first job in WQ* (FJ) – the first job in WQ.

An allocation event is triggered when a new job is submitted to the system or when a running job finishes its execution. For each allocation event the system tries to continuously allocate as many jobs as possible. It stops allocation only when there are no sites being able to accommodate the first job in the waiting queue or when the waiting queue becomes empty. The proposed *intelligent* method is designed to dynamically adjust the allocation method between *best-fit* and *fastest-first* whenever making allocation decision.

Not every triggered allocation event leads to actual allocation results since there might be no enough resources or no jobs to allocate. Table II classifies all possible allocation events into four types of situations according to the status of waiting queue and the causes that trigger the events. The symbol "X" represents that there will be no actual allocation in that situation. Since we apply FCFS as the scheduling policy, $Size_{WQ} > 0$ implies that there is no site being able to accommodate the first job in waiting queue and that the newly submitted job must wait in the rear of waiting queue. For the case that $Size_{WQ} = 0$ and the triggering event is job finish, there are no jobs to allocate and therefore no actual allocation happens. Only situations (a) and (b) in Table II would lead to actual allocation results if there is any site which can accommodate the submitted job or the first job in waiting queue.

In situation (a), $Size_{WQ} = 0$ implies the system loading is low so it comprises the (low, low) or (high, low) situation mentioned in the previous section. For the (high, low) case, we know that *fastest-first* is a potentially better choice. Thus we compare the computing speeds of the selected sites with different allocation methods to make the allocation decision. The allocation decision is determined by equation (1).

$$\text{Final Decision} = \begin{cases} \textit{best-fit}, & \text{if } CS_{S_{bf}(FJ)} \geq CS_{S_{ff}(FJ)} \\ \textit{fastest-first}, & \text{if } CS_{S_{bf}(FJ)} < CS_{S_{ff}(FJ)} \end{cases} \quad (1)$$

In situation (b), which comprised the (low, high) or (high, high) case, we make the allocation decision by calculating which allocation method can allow subsequent jobs in waiting queue to consume more computing capacity. The computing capacity $CC(i)$ consumed by job i is defined as

$$CC(i) = \begin{cases} CS_j \times RNP_i, & \text{where job i is allocated to site j.} \\ 0, & \text{where job i can not be allocated to any site.} \end{cases} \quad (2)$$

$CC_{bf}(i)$ and $CC_{ff}(i)$ are used to denote that job i is allocated by *best-fit* and *fastest-first* respectively. Thus the total computing capacity consumed by *best-fit* and *fastest-first* are denoted by TCC_{bf} and TCC_{ff} respectively and defined as

$$TCC_{ff} = CC_{ff}(FJ) + \textstyle\sum_{\forall i \text{ in WQ exclude FJ}} CC_{bf}(i) \quad (3)$$

$$TCC_{bf} = \textstyle\sum_{\forall i \text{ in WQ}} CC_{bf}(i) \quad (4)$$

A value *Score* which represents the relative performance of *best-fit* and *fastest-first* is then calculated by

$$Score = \frac{CS_{S_{ff}(FJ)}}{CS_{S_{bf}(FJ)}} \times \frac{TCC_{ff}}{TCC_{bf}} \quad (5)$$

The allocation decision for situation (b) is then determined by equation (6).

$$\text{Final Decision} = \begin{cases} \textit{fastest-fast}, & \text{if Score} > 1 \\ \textit{best-fit}, & \text{if Score} \leq 1 \end{cases} \quad (6)$$

The proposed *intelligent* allocation method is inspired by the *adaptive* allocation strategy presented in [9] which makes allocation decision based on a calculation of which policy can further accommodate more jobs for immediate execution. The improvement in the *intelligent* allocation method is to take the speed difference into account. The pseudo code of the *intelligent* allocation algorithm is shown in Fig. 1.

TABLE II. CLASSIFICATION OF ALLOCATION EVENTS

	Submit	Finish
$Size_{WQ} = 0$	(a)	X
$Size_{WQ} > 0$	X	(b)

```
Algorithm IntelligentAllocator()
{
  calculate S_bf(FJ) and S_ff(FJ)
  if S_bf(FJ) = S_ff(FJ)
    choose the site suggested by both methods
  end if
  if Size_WQ = 0 and event = Submit
    if CS_S_bf(FJ) ≥ CS_S_ff(FJ)
      choose best-fit
    otherwise
      choose fastest-first
    end if
  end if
  calculate Score
  if Score > 1
    choose fastest-first
  otherwise
    choose best-fit
  end if
}
```

Figure 1. Pseudo code of the proposed *intelligent* allocation algorithm

III. EXPERIMENTS AND DISCUSSIONS

A. Performance Metrics and Experimental Settings

Our simulation studies were based on publicly downloadable workload traces [11]. We used the SDSC's

SP2 workload logs on [11] as the basic input workload in the following simulations. Other workloads for simulating different workload conditions were derived from the basic workload. We used the average response time (*AverageResponseTime*) of all jobs as the performance metric to compare different allocation methods in all simulations. The *AverageResponseTime* is defined by

$$AverageResponseTime = \frac{\sum_{j \in AllJobs} endTime_j - submitTime_j}{TotalNumberofJobs}$$

We compared the proposed *Intelligent* (IT) method with the *adaptive* (AD) [9], *best-fit* (BF) [5], and *fastest-first* (FF) [9] methods. In order to evaluate the performance of the proposed method on various workload conditions, we conducted a series of experiments by varying three adjustable parameters listed in Table III. Speed heterogeneity (SH), represented by the variance of computing speeds of all sites, ranges from 0 to 0.2. For better understanding of the influence of SH, setting SH = 0.05, 0.01, 0.15, and 0.2 will averagely make the speed of the fastest site 1.8, 2.3, 3, and 4.2 times faster than the speed of the slowest site respectively. SH=0 reduces to the homogeneous case. We randomly generate 10 sets of speed setting with respect to each SH value. All presented experimental results are the average value of these 10 sets.

System loading (SL), ranging from 1 to 5, is simulated by multiplying the execution time of each job with the corresponding value (e.g., SL = 2 doubles the execution time of all jobs). The following uses the average length of waiting queue for homogeneous case (SH = 0) and the *best-fit* method as an example to show the effect of SL. The length of waiting queue will be 0.9, 7.8, 98, 2618, and 6717 as SL is set to 1, 2, 3, 4, and 5 respectively.

Resource configuration (RC) defined by

$$RC = \frac{\mathrm{Max}(\mathrm{RNP_i}), \text{for all job i in simulation}}{\mathrm{Max}(\mathrm{NP_j}), \text{for all sites j}}$$

ranges from 100% to 25% with a step of 25% in the simulations. In the SDSC's SP2 system the jobs in the log were put into five different queues. With RC = 100%, we use the maximum number of requested processors of all jobs in each queue as the size of each site, which were 8, 128, 128, 128, 50 corresponding to queue 1 to queue 5 respectively. This resource setting was used for all simulations. For other RC settings, we simulated it by cutting a job that exceeds the specified percentage into several small jobs. For example, when RC = 25%, a job requesting 100 processors was cut into four small jobs, where three of them each requested 32 processors (128 × 25%) and the last one asked for the remaining 4 processors. Table IV shows the characteristics of SDSC's SP2 workload with respect to different RC settings.

Note that only SL will change the amount of workload brought into the system while the other two parameters neither change the total computing capability of all resources nor change the average workload brought into the system.

TABLE III. PARAMETERS FOR EXPERIMENTS

Resource Heterogeneity (SH)	{0, 0.05, 0.1, 0.15, 0.2}
System Loading (SL)	{1, 2, 3, 4, 5}
Resource Configuration (RC)	{100%, 75%, 50%, 25%}

TABLE IV. CHARACTERISTIC OF SDSC'S SP2 WORKLOAD WITH RESPECT TO DIFFERENT RC SETTINGS

	Number of jobs	Maximum number of processors per job	Average number of requested processors per job
RC=100%	54041	128	12.29
RC=75%	54305	96	12.23
RC=50%	54534	64	12.18
RC=25%	58890	32	11.28

B. Experimental Results and discussion

Fig. 2 shows the performance of each allocation methods in terms of *AverageResponseTime*. Each sub-figure shows the simulation result performed by varying the SH form 0 to 0.2 with specific SL and RC setting. For simplicity and clarity, we only show the results of SL from 2 to 4. The results of SL = 1 and SL = 5 actually follow the same performance trend.

From all the sub-figures we can observe that in the (low, high) case (see sub-figures (c), (f), (i), and (l) with SH = 0 and 0.05) *best-fit* surpasses *fastest-first*. This observation is consistent with our analysis in Table I. The experimental results in sub-figures (a), (d), (g), and (j) also confirm another analysis in Table I, which indicates that *fastest-first* outperforms *best-fit* for case (high, low). Moreover, these results show that no single existing processor allocation method can always perform the best under all possible workload conditions.

For the performance of the proposed *intelligent* method, we calculated how many times it is the best method or close to the best method in all 100 parameter settings (5 × 5 × 4 = 100), as show in Table V. The performances of two allocation methods are said to be close to each other if the difference ratio of *AverageResponseTime* is less than 1%. The result shows that in 39 of 100 cases the proposed *intelligent* method performed better than all other methods and in other 37 of 100 cases it is close to the best allocation method. This result demonstrates that the proposed *intelligent* allocation method can dynamically adapt to various workload conditions and thus deliver better performance in average.

Comparing the *intelligent* and the *adaptive* methods also finds that the *intelligent* method surpasses the *adaptive* method in 64 of 100 cases.

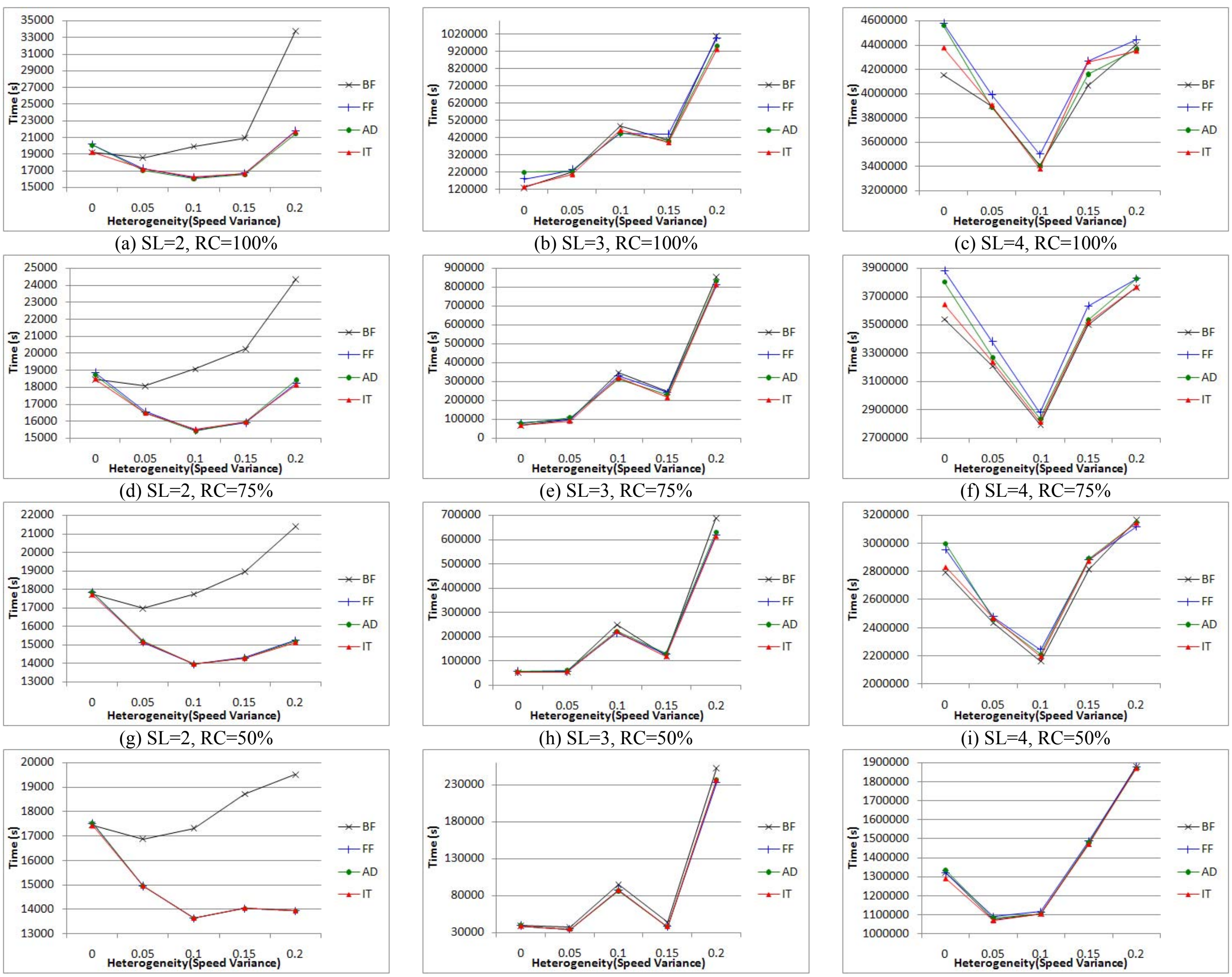
TABLE V. THE NUMBER OF TIMES THE *INTELLIGENT* METHODS IS THE BEST OR CLOSE TO THE BEST ALOCATION METHOD

	intelligent is the best method	*intelligent* is close to the best method	Summary
RC=100%	6/25	10/25	16/25
RC=75%	12/25	9/25	21/25
RC=50%	12/25	5/25	17/25
RC=25%	9/25	13/25	22/25
Total	39/100	37/100	76/100

IV. CONCLUSTIONS

For heterogeneous grid environments, no existing processor allocation methods can consistently deliver the best performance under different resource and workload conditions. Moreover, some of these workload conditions change with user behavior that is hard to predict in advance when system administrator decides which allocation method to be used. Thus no performance guarantee could be made. This paper analyzes the relative strength of existing allocation methods and presents an *intelligent* processor allocation method, which improves system performance through considering both effects of the speed heterogeneity and resource fragmentation. Extensive simulation studies have been conducted to evaluate the proposed method. The experimental results show that the proposed *intelligent* method can dynamically adapt to the better allocation method between *best-fit* and *fastest-first*. Therefore, it can effectively deliver better performance under most workload and resource conditions.

It is difficult to develop a processor allocation method which can always perform the best under all possible conditions. In addition to the proposed method, the extensive simulation analysis of different allocation methods under various conditions in this paper can serve as a good basis for better understanding of the root causes of the performance difference between the methods. The understanding could in turn help develop more effective processor allocation methods.



(a) SL=2, RC=100% (b) SL=3, RC=100% (c) SL=4, RC=100%

(d) SL=2, RC=75% (e) SL=3, RC=75% (f) SL=4, RC=75%

(g) SL=2, RC=50% (h) SL=3, RC=50% (i) SL=4, RC=50%

(j) SL=2, RC=25% (k) SL=3, RC=25% (l) SL=4, RC=25%

Figure 2. AverageResponseTime of the *best-fit*, *fastest-first*, *adaptive*, and *Intelligent* methods with various SH, SL, and RC settings

ACKNOWLEDGMENT

This paper is based upon work supported by National Science Council (NSC), Taiwan, under grants no. NSC 96-2221-E-007-130-MY3, NSC 97-3114-E-007-001, and NSC 96-2221-E-432-003-MY3. The authors also thank all of the people for comments and advices.

REFERENCES

[1] C. Ernemann, V. Hamscher, R. Yahyapour, “Benefits of Global Grid Computing for Job Scheduling,” Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing(GRID’04), pp. 374-379, November 2004.

[2] L. M. Ni, S. W. Turner, B. H. C. Cheng, "Contention-Free 2D-Mesh Cluster Allocation in Hypercubes," IEEE Transactions on Computers, vol. 44, no. 8, pp. 1051-1055, Aug. 1995.

[3] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, Inc., 1999.

[4] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, “Evaluation of Job-Scheduling Strategies for Grid Computing”, Proceedings of the 7th International Conference on High Performance Computing, HiPC-2000, pp. 191-202, Bangalore, India, 2000.

[5] K. C. Huang and H. Y. Chang, “An Integrated Processor Allocation and Job Scheduling Approach to Workload Management on Computing Grid”, Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'06), pp. 703-709, Las Vegas, USA, June 26-29, 2006.

[6] D. G. Feitelson, and L. Rudolph, “Parallel Job Scheduling: Issues and Approaches”, Proceedings of IPPS’95 Workshop: Job Scheduling Strategies for Parallel Processing, pp. 1-18, 1995.

[7] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, “Theory and Practice in Parallel Job Scheduling”, Job Scheduling Strategies for Parallel Processing, pp. 1-34, Springer-Verlag, 1997.

[8] D. D. Sharma, D. K. Pradhan, "Processor Allocation in Hypercube Multicomputers: Fast and Efficient Strategies for Cubic and Noncubic Allocation," IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 10, pp. 1108-1122, Oct. 1995.

[9] K. C. Huang, P. C. Shih, Y. C. Chung, “Towards Feasible and Effective Load Sharing in a Heterogeneous Computational Grid”, Proceedings of the Second International Conference on Grid and Pervasive Computing, France (2007)

[10] A. W. Mu’alem, D. G. Feitelson, “Utilization, Predictability, Workloads, and User Runtime Estimate in Scheduling the IBM SP2 with Backfilling”, IEEE Transactions on Parallel and Distributed Systems, Vol. 12, Iss. 6, pp. 529-543, 2001.

[11] Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/

[12] O. H. Kwon, J. Kim, S. J. Hong, S. G. Lee, "Real-Time Job Scheduling in Hypercube Systems," Proceedings of 1997 International Conference on Parallel Processing (ICPP '97), pp.166, 1997.

[13] D. Lifka, “The ANL/IBM SP Scheduling System,” Proc. Int'l Parallel and Distributed Processing Symp. Workshop Job Scheduling Strategies for Parallel Processing vol. 949, pp. 295-303, Apr. 1995.

[14] J. Skovira, W. Chan, H. Zhou, and D. Lifka, “The EASY-LoadLeveler API Project,” Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., pp. 41–47, 1996.

[15] O. H. Kwon, K. Y. Chwa, "An Algorithm for Scheduling Jobs in Hypercube Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 9, pp. 856-860, Sept. 1998.