Adaptive Processor Allocation for Moldable Jobs in Computational Grid

Kuo-Chan Huang, National Taichung University, Taiwan

Po-Chi Shih, National Tsing Hua University, Taiwan

Yeh-Ching Chung, National Tsing Hua University, Taiwan

ABSTRACT

In a computational grid environment, a common practice is try to allocate an entire parallel job onto a single participating site. Sometimes a parallel job, upon its submission, cannot fit in any single site due to the occupation of some resources by running jobs. How the job scheduler handles such situations is an important issue which has the potential to further improve the utilization of grid resources as well as the performance of parallel jobs. This paper develops adaptive processor allocation policies based on the moldable property of parallel jobs to deal with such situations in a heterogeneous computational grid environment. The proposed policies are evaluated through a series of simulations using real workload traces. The results indicate that the proposed adaptive processor allocation policies can further improve the system performance of a heterogeneous computational grid significantly.

Keywords: adaptive processor allocation; computational grid; job scheduling; moldable property

INTRODUCTION

Most parallel computing environments running scientific applications adopt the space-sharing approach. In this approach, the processing elements of a parallel computer are logically partitioned into several groups. Each group is dedicated to a single job, which may be serial or parallel. Therefore, each job has exclusive use of the group of processing elements allocated to it when it is running. However, different running jobs may have to share the networking and storage resources to some degree.

Most current parallel application programs have the *moldable* property (Dror, Larry, Uwe, Kenneth and Parkson, 1997). It means the programs are written in a way so that at runtime they can exploit different parallelisms for execution according to specific needs or available resource. Parallelism here means the number of processors a job uses for its execution. The *moldable* property raises an interesting ques-

tion whether it is possible to design special processor allocation policies, taking advantage of this property, to improve the overall system performance.

This paper develops adaptive processor allocation policies based on the moldable property of parallel jobs for both homogeneous parallel computers and heterogeneous computational grid environments. The proposed policies require users to provide estimations of job execution times upon job submission. The policies are evaluated through a series of simulations using real workload traces. The effects of inexact runtime estimations on system performance are also investigated. The results indicate that the proposed adaptive processor allocation policies are effective as well as stable under different system configurations and can tolerate a wide range of estimation errors.

RELATED WORK

This paper deals with scheduling and allocating independent parallel jobs in a heterogeneous computational grid. Without grid computing, local users can only run jobs on the local site. The owners or administrators of different sites are interested in the consequences of participating in a computational grid, whether such participation will result in better service for their local users by improving the job turnaround time. A common load-sharing practice is to allocate an entire parallel job to a single site which is selected from all sites in the grid based on some criteria. However, sometimes a parallel job, upon its submission, cannot fit in any single site due to the occupation of some resources by running jobs. How the job scheduler handles such situations is an important issue which has the potential to further improve the utilization of grid resources as well as the performance of parallel jobs.

Job scheduling for parallel computers has been subject to research for a long time. As for grid computing, previous works discussed several strategies for a grid scheduler. One approach is the modification of traditional list scheduling strategies for usage on grid (Carsten, Volker, Uwe, Ramin and Achim, 2002; Carsten Ernemann, Hamscher, Streit and Yahyapour, 2002a, 2002b; Hamscher, Schwiegelshohn, Streit and Yahyapour, 2000). Some economic based methods are also being discussed (Buyya, Giddy, & Abramson, 2000; Carsten, Volker and Ramin, 2002; Rajkumar Buyya, 2002; Yanmin et al., 2005). In this paper, we explore non economic scheduling and allocation policies with support for a speed-heterogeneous grid environment.

England and Weissman (2005) analyzed the costs and benefits of load sharing of parallel jobs in the computational grid. Experiments were performed for both homogeneous and heterogeneous grids. However, in their works simulations of a heterogeneous grid only captured the differences in capacities and workload characteristics. The computing speeds of nodes on different sites are assumed to be identical. In this paper, we deal with load sharing issues regarding heterogeneous grids in which nodes on different sites may have different computing speeds.

For load sharing there are several methods possible for selecting which site to allocate a job. Earlier simulation studies in the literature (Hamscher et al., 2000; Huang and Chang, 2006) showed the best results for a selection policy called *best-fit*. In this policy a particular site is chosen on which a job will leave the least number of free processors if it is allocated to that site. However, these simulation studies are performed based on a computational grid model in which nodes on different sites all run at the same speed. In this paper we explore possible site selection policies for a heterogeneous computational grid. In such a heterogeneous environment nodes on different sites may run at different speeds.

In the literature (Barsanti and Sodan, 2007; John, Uwe, Joel and Philip, 1994; Sabin, Lang, and Sadayappan, 2007; Srividya, Vijay, Rajkumar, Praveen and Sadayappan, 2002; Sudha, Savitha and Sadayappan, 2003; Walfredo and Francine, 2000, 2002) several strategies for scheduling moldable jobs have been introduced.

Most of the previous works either assume the job execution time is a known function of the number of processors allocated to it or require users to provide estimated job execution time. In Huang (2006) without the requirement of known job execution time three adaptive processor allocation policies for moldable jobs were evaluated and shown to be able to improve the overall system performance in terms of average job turnaround time. Most of the previous work deals with scheduling moldable jobs in a single parallel computer or in a homogeneous grid environment. In this paper, we explore adaptive processor allocation in a heterogeneous computational grid environment.

COMPUTATIONAL GRID MODEL AND EXPERIMENTAL SETTING

In this section, the computational grid model is introduced on which the evaluations of the proposed policies are based. In the model, there are several independent computing sites with their own local workload and management system. This paper examines the impact on performance results if the computing sites participate in a computational grid with appropriate job scheduling and processor allocation policies. The computational grid integrates the sites and shares their incoming jobs. Each participating site is a homogeneous parallel computer system. The nodes within each site run at the same speed and are linked with a fast interconnection network that does not favor any specific communication pattern (Feitelson and Rudolph, 1995). This means a parallel job can be allocated on any subset of nodes in a site. The parallel computer system uses space-sharing and run the jobs in an exclusive fashion.

The system deals with an on-line scheduling problem without any knowledge of future job submissions. The jobs under consideration are restricted to batch jobs because this job type is dominant on most parallel computer systems running scientific and engineering applications. For the sake of simplicity, in this paper we assume a global grid scheduler which handles all job scheduling and resource allocation activities. The local schedulers are only responsible for starting the jobs after their allocation by the global scheduler. Theoretically, a single central scheduler could be a critical limitation concerning efficiency and reliability. However, practical distributed implementations are possible, in which site-autonomy is still maintained but the resulting schedule would be the same as created by a central scheduler (Ernemann, Hamscher and Yahyapour, 2004).

For simplification and efficient load sharing all computing nodes in the computational grid are assumed to be binary compatible. The grid is heterogeneous in the sense that nodes on different sites may differ in computing speed and different sites may have different numbers of nodes. When load sharing activities occur a job may have to migrate to a remote site for execution. In this case the input data for that job have to be transferred to the target site before the job execution while the output data of the job is transferred back afterwards. This network communication is neglected in our simulation studies as this latency can usually be hidden in pre- and post-fetching phases without regards to the actual job execution phase (Ernemann et al., 2004).

In this paper, we focus on the area of high throughput computing, improving system's overall throughput with appropriate job scheduling and allocation methods. Therefore, in our studies, the requested number of processors for each job is bound by the total number of processors on the local site from which the job is submitted. The local site in which a job is submitted from will be called the *home site* of the job henceforth in this paper. We assume all jobs have the moldable property. It means the programs are written in a way so that at runtime they can exploit different parallelisms for execution according to specific needs or available resource. Parallelism here means the number of processors a job uses for its execution. In our model we associated each job with several attributes. The following five attributes

are provided before a simulation starts. The first four attributes are directly gotten from the SDSC SP2's workload log. The *estimated runtime* attribute is generated by the simulation program according to the specified range of estimation errors and their corresponding statistical distributions.

- **Site number**. This indicates the home site of a job which it belongs to.
- **Number of processors**. It is the number of processors a job uses according to the data recorded in the workload log.
- **Submission time**. This provides the information about when a job is submitted to its home site.
- **Runtime**. It indicates the required execution time for a job using the specified number of processors on its home site. This information for runtime is required for driving the simulation to proceed.
- **Estimated runtime.** An estimated runtime is provided upon job submission by the user. The job scheduler uses this information to guide the determination process of job scheduling and allocation.

The following job attributes are collected and calculated during the simulation for performance evaluation.

- **Waiting time**. It is the time between a job's submission and its allocation.
- **Actual runtime**. When adaptive processor allocation is applied to a job, its actual runtime may be different from the runtime recorded in the workload log. This attribute records the actual runtime it takes.
- **Actual number of processors**. When the scheduler applies adaptive processor allocation to a job, the number of processors the job actually uses for execution may be different from the value recorded in the workload log. This attribute records the number of processors actually used.
- **Execution site**. In a computational grid environment, a job may be scheduled to run on a site other than its home site. The attribute records the actual site that it runs on.
- **Turnaround time**. The simulation program calculates each job's turnaround time after its execution and records the value in this attribute.

Our simulation studies were based on publicly downloadable workload traces ("Parallel Workloads Archive,"). We used the SDSC's SP2 workload logs on ("Parallel Workloads Archive,") as the input workload in the simulations. The detailed workload characteristics are shown in Table 1.

In the SDSC's SP2 system the jobs in the logs are put into different queues and all these queues share the same 128 processors. In section 4, this original workload is directly used to simulate a homogeneous parallel computer with 128 processors. In section 5, the work-

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				

load log will be used to model the workload on a computational grid consisting of several different sites whose workloads correspond to the jobs submitted to the different queues respectively. Table 2 shows the configuration of the computational grid according to the SDSC's SP2 workload log. The number of processors on each site is determined according to the maximum number of required processors of the jobs belonged to the corresponding queue for that site.

To simulate the speed difference among participating sites we define a speed vector, *e.g.* speed=(sp1,sp2,sp3,sp4,sp5), to describe the relative computing speeds of all the five sites in the grid, in which the value 1 represents the computing speed resulting in the job execution time in the original workload log. We also define a load vector, *e.g.* load=(ld1,ld2,ld3,ld4,ld5), which is used to derive different loading levels from the original workload data by multiplying the load value ld_i to the execution times of all jobs at site *i*.

ADAPTIVE PROCESSOR ALLOCATION ON HOMOGENEOUS PARALLEL COMPUTER

The *moldable* property raises an interesting question whether it is possible to design special processor allocation policies, taking advantage of this property, to improve the overall system performance. For example, an intuitive idea is allowing a job to use a less number of processors than originally specified for immediate execution if at that moment the system has not enough free processors otherwise the job has to wait in a queue for a uncertain period of time. On the other hand, if the system has more free processors than a job's original requirement, the system might let the job to run with more processors than originally required to shorten its execution time. This is called *adaptive processor allocation* in this paper. Therefore, the system can dynamically determine the runtime parallelism of a job before its execution through adaptive processor allocation to improve system utilization or reduce the job's waiting time in queue.

For a specific job, intuitively we know that allowing higher parallelism can lead to shorter execution time. However, when the overall system performance is concerned, the positive effects of raising a job's parallelism can not be so assured under the complex system behavior. For example, although raising a job's parallelism can reduce its required execution time, it might, however, increase other jobs' probability of having to wait in queue for longer time. This would increase those jobs' waiting time and in turn turnaround time. Therefore, it is not straightforward to know how raising a single job's parallelism would affect the overall system-level performance, *e.g.* the average turnaround time of all jobs. On the other hand, reducing a job's parallelism might shorten its waiting time in queue at the cost of enlarged execution time. It is not always clear whether the combined effects of shortened waiting time and enlarged execution time would lead to a reduced or increased overall turnaround time. Moreover, the reduced parallelism of a job would usually in turn result in the decreased waiting time of other jobs. This makes it even more complex to analyze the overall system effects.

The above examples illustrate that the effects of the idea of adaptive processor allocation on overall system performance is complex

Table 2. Configuration of the computational grid according to SDSC's SP2 workload

	total	site 1	site 2	site 3	site 4	site 5
Number of processors	442	8	128	128	128	50

and require further evaluation. In our previous work (Huang, 2006), we proposed two possible adaptive processor allocation policies. In this paper, we improve the two policies by requiring users to provide estimated job execution time upon job submission, just like what is required by the backfilling algorithms. The estimated job execution time is used to help the system determine whether to dynamically scale down a job's parallelism for immediate execution, *i.e.* shorter waiting time, at the cost of longer execution time or to keep it waiting in queue for the required amount of processors to become available. This section explores and evaluates the two improved adaptive processor allocation policies which take advantage of the *moldable* property on homogeneous parallel computers. The three processor allocation policies to be evaluated are described in detail in the following.

- **No adaptive scaling.** This policy allocates the number of processors to each parallel job exactly according to its specified requirement. The policy is used in this section as the performance basis for evaluating the adaptive processor policies.
- **Adaptive scaling down.** If a parallel job specifies an amount of processors which at that moment is larger than the number of free processors. The system has two choices for scheduling the job: scaling its parallelism down for immediate execution or keeping it waiting in queue. According to the estimated execution time of the job, the system can compute the job's enlarged execution time once scaling down its parallelism. On the other hand, based on the estimated execution time of each job running on the system, it is possible to predict how long it will take for the system to gather enough free processors to fulfill the original requirement of the job. Therefore, the system can compare the resultant performances of the two choices and choose the better one. We use a threshold variable to control the selection between the two choices. The system chooses to scale down the job's parallelism for immediate execution only if, where To is the predicted turnaround time if the job waits in queue until enough free processors are available and Tsd is the predicted turnaround time if the job run immediately with reduced parallelism.
- **Conservative scaling up and down.** In addition to the scaling down mechanism described in the previous policy, this policy automatically scales a parallel job's parallelism up to use the amount of total free processors even if its original requirement is not that large. However, to avoid a single job from exhausting all free processors, resulting in subsequent jobs' unnecessary enlarged waiting time in queue, the policy scales a parallel job's parallelism up only if there are no jobs behind it in queue. This is why it is called *conservative*.

Table 3 shows the performance evaluation of various processor allocation policies. For the adaptive policies with runtime estimation, we

Table 3. Performance comparison of adaptive processor allocation policies

	Average turnaround time (sec.)
No adaptive scaling	51677
Adaptive scaling down (without runtime estimation)	17805
Adaptive scaling down (with runtime estimation)	13746
Conservative scaling up and down (without runtime estimation)	18072
Conservative scaling up and down (with runtime estimation)	12270

experimented with several possible threshold values and chose the best result to present in Table 3. For the adaptive scaling down policy, the best threshold value is 2.1 and the conservative scaling up and down policy delivers the best performance when the threshold value is 2. Table 3 shows that adaptive processor allocation in general can improve the overall system performance several times, compared to the traditional allocation policy sticking to a job's original amount of processor requirement. Moreover, the improved adaptive processor allocation policies presented in this paper can further improve the performance significantly with the aid of runtime estimation. For the original adaptive allocation policies, allowing scaling up parallelism cannot improve system performance further in addition to scaling down parallelism in terms of average turnaround time. However, for the improved adaptive allocation policies, scaling up parallelism does improve the system performance delivered by the policy which scales down the parallelism only. Overall speaking, the conservative scaling up and down policy with runtime estimation outperforms the other policies.

The studies in Table 3 assume that users always provide exact estimations of job execution times. However, this is by no means possible in real cases. Therefore, we performed additional simulation studies to evaluate the stability of the adaptive processor allocation policies when users provide only inexact estimations. The results are presented in Table 4. The error range of estimation is relative to a job's actual execution time. Table 4 shows that sometimes small estimation error might even lead to better performance than exact estimation such as the case of conservative scaling up and down with a 20% error range. In general, a larger error range results in degraded performance. However, up to 90% error range, the improved adaptive policies with runtime estimation still outperform the original adaptive policies, compared to Table 3. The results illustrate that the proposed adaptive processor allocation policies are stable and practical.

Table 4. Effects of inexact runtime estimation under uniform distribution

	Average turnaround time (sec.)
Adaptive scaling down	
0% error range of estimation	13746
20% error range of estimation	14217
40% error range of estimation	14660
60% error range of estimation	14934
80% error range of estimation	15264
90% error range of estimation	15582
Conservative scaling up and down	
0% error range of estimation	12270
20% error range of estimation	11926
40% error range of estimation	12396
60% error range of estimation	12607
80% error range of estimation	14657
90% error range of estimation	17964

The simulations for Table 4 assume the estimation errors conform to the uniform distribution. In the following, Table 5 presents another series of simulations which evaluate the cases where the estimation errors conform to the normal distribution. The results again show that sometimes larger error ranges lead to better performances. Moreover, Table 5 indicates that the adaptive processor allocation policies perform even more stably under the normal distribution of estimation errors, compared to Table 4.

ADAPTIVE PROCESSOR ALLOCATION IN HETEROGENEOUS GRID

In a computational grid environment, a common practice is try to allocate an entire parallel job onto a single participating site. Sometimes a parallel job, upon its submission, cannot fit in any single site due to the occupation of some processors by running jobs. How the job scheduler handles such situations is an important issue which has the potential to further improve the utilization of grid resources as well as the performance of parallel jobs. This section extends the adaptive processor allocation policies proposed in the previous sections to deal with such situations in a heterogeneous computational grid environment.

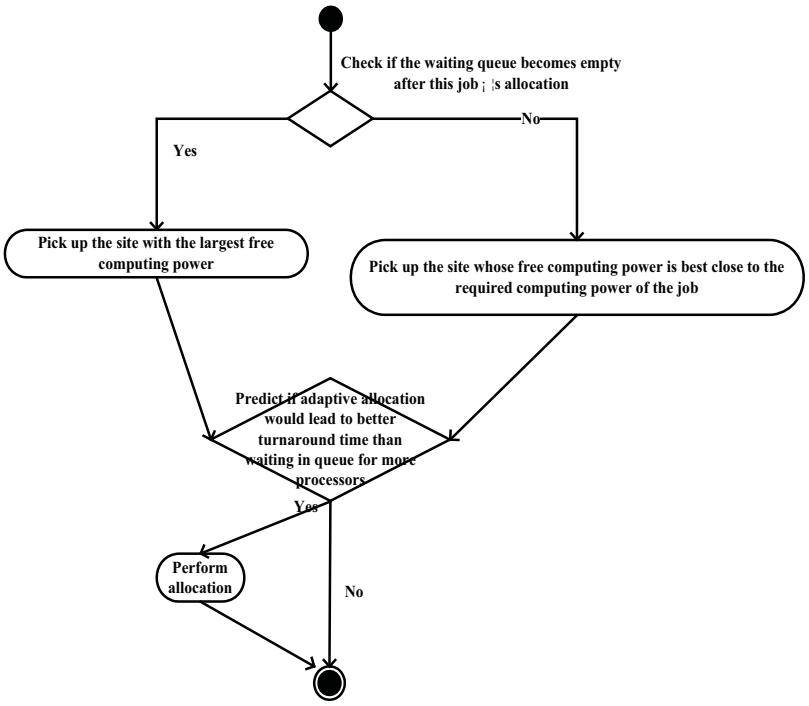
The detailed adaptive processor allocation procedure is illustrated in Figure 1. The major difference between the adaptive processors allocation procedures for a homogeneous parallel computer and for a heterogeneous grid environment is the site selection process regarding the computation and comparison of computing power of different sites. A site's free computing power is defined as the number of free processors on it multiplied by the computing speed of a single processor. Similarly, the required computing power of a job is defined as the number of required processors specified in the job multiplied by the computing speed of a single processor on its home site.

In the following, we compare the performances of five different cases. They are

Table 5. Effects of inexact runtime estimation under normal distribution

	Average turnaround time (sec.)
Adaptive scaling down	
0% error range of estimation	13746
20% error range of estimation	13958
40% error range of estimation	14067
60% error range of estimation	13897
80% error range of estimation	14217
100% error range of estimation	14068
Conservative scaling up and down	
0% error range of estimation	12270
20% error range of estimation	12491
40% error range of estimation	12310
60% error range of estimation	11816
80% error range of estimation	12350
90% error range of estimation	12895

Figure 1. Adaptive processor allocation procedure in heterogeneous grid



independent clusters representing a non-grid architecture, adaptive processor allocation without runtime estimation, adaptive processor allocation with exact runtime estimation, adaptive processor allocation with uniform distribution of runtime-estimation errors, adaptive processor allocation with normal distribution of runtime-estimation errors. Table 6 presents the results of simulations for a heterogeneous computational grid with speed vector (1,3,5,7,9) and load vector (10,10,10,10,10). For the last two cases in Table 6, we present their worst-case data within the estimation-error range from 10% to 100% with the step of 10%. The results in Table 6 show that grid computing with adaptive processor allocation can greatly improve the system performance compared to the non-grid architecture. Moreover, the improved adaptive processor allocation policies with runtime estimation can improve the system performance further compared to the original adaptive processor allocation policy. The results also indicate that estimation errors lead to little influence on overall system performance. Therefore, the proposed adaptive allocation policies are stable in a heterogeneous computational grid.

Table 6 represents only one possible speed configuration in a heterogeneous computational grid environment. To further investigate the effectiveness of the proposed policies, we conducted a series of 120-case simulations corresponding to all possible permutations of the site speed vector (1,3,5,7,9) under the SDSC's SP2 workload. Table 7 shows the average turnaround times over the 120 cases for the five allocation policies in Table 6, accordingly. The results again confirm that the proposed adaptive processor allocation policies are stable and can significantly improve system performance. For the details, among all the 120 cases, the proposed adaptive allocation policies

Table 6. Performance evaluation in a heterogeneous computational grid

	Average turnaround time (sec.)
Independent clusters	51163
Adaptive processor allocation without runtime estimation	17381
Adaptive processor allocation with exact runtime estimation	15953
Adaptive processor allocation with uniform distribution of runtime-estimation errors	15956
adaptive processor allocation with normal distribution of runtime-estimation errors	15953

Table 7. Average performance over 120 different speed configurations

	Average turnaround time (sec.)
Independent clusters	8562757
Adaptive processor allocation without runtime estimation	20987
Adaptive processor allocation with exact runtime estimation	19695
Adaptive processor allocation with uniform distribution of runtime-estimation errors	19732
adaptive processor allocation with normal distribution of runtime-estimation errors	19720

with runtime estimation outperform the original adaptive policy in 108 cases.

CONCLUSION

In the real world, a grid environment is usually heterogeneous at least for the different computing speeds at different participating sites. The heterogeneity presents a challenge for effectively arranging load sharing activities in a computational grid. This paper develops adaptive processor allocation policies based on the moldable property of parallel jobs for heterogeneous computational grids. The proposed policies can be used when a parallel job, during the scheduling activities, cannot fit in any single site in the grid. The proposed policies require users to provide estimations of job execution times upon job submission. The policies are evaluated through a series of simulations using real workload traces. The results indicate that the adaptive processor allocation policies can further improve the system performance of a heterogeneous computational grid significantly when parallel jobs have the moldable property. The effects of inexact runtime estimations on system performance are also investigated. The results indicate that the proposed adaptive processor allocation policies are effective as well as stable under different system configurations and can tolerate a wide range of estimation errors.

REFERENCE

Barsanti, L., & Sodan, A. (2007). Adaptive Job Scheduling Via Predictive Job Resource Allocation. *Job Scheduling Strategies for Parallel Processing* (pp. 115-140).

Buyya, R., Giddy, J., & Abramson, D. (2000). *An Evaluation of Economy-Based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications.* Paper presented at the Proceedings of the Second Workshop on Active Middleware Services (AMS2000), Pittsburgh, USA.

Carsten, E., Volker, H., & Ramin, Y. (2002). *Economic Scheduling in Grid Computing*. Paper presented at the Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing.

Carsten, E., Volker, H., Uwe, S., Ramin, Y., & Achim, S. (2002). *On Advantages of Grid Computing for Parallel Job Scheduling*. Paper presented at the Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid.

Dror, G. F., Larry, R., Uwe, S., Kenneth, C. S., & Parkson, W. (1997). *Theory and Practice in Parallel Job Scheduling*. Paper presented at the Proceedings of the Job Scheduling Strategies for Parallel Processing.

England, D., & Weissman, J. B. (2005). Costs and Benefits of Load Sharing in the Computational Grid. In *Job Scheduling Strategies for Parallel Processing* (pp. 160-175).

Ernemann, C., Hamscher, V., Streit, A., & Yahyapour, R. (2002a). Enhanced Algorithms for Multi-site Scheduling. In *Grid Computing — GRID 2002* (pp. 219-231).

Ernemann, C., Hamscher, V., Streit, A., & Yahyapour, R. (2002b). On Effects of Machine Configurations on Parallel Job Scheduling in Computational Grids. *Proceedings of International Conference on Architecture of Computing Systems, ARCS*, 169-179.

Ernemann, C., Hamscher, V., & Yahyapour, R. (2004). *Benefits of global grid computing for job scheduling.* Paper presented at the Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on.

Feitelson, D., & Rudolph, L. (1995). Parallel job scheduling: Issues and approaches. In *Job Scheduling Strategies for Parallel Processing* (pp. 1-18).

Hamscher, V., Schwiegelshohn, U., Streit, A., & Yahyapour, R. (2000). Evaluation of Job-Scheduling Strategies for Grid Computing. In *Grid Computing — GRID 2000* (pp. 191-202).

Huang, K.-C. (2006). *Performance Evaluation of Adaptive Processor Allocation Policies for Moldable Parallel Batch Jobs*. Paper presented at the Proceedings of the Third Workshop on Grid Technologies and Applications.

Huang, K.-C., & Chang, H.-Y. (2006). *An Integrated Processor Allocation and Job Scheduling Approach to Workload Management on Computing Grid.* Paper presented at the Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'06), Las Vegas, USA.

John, T., Uwe, S., Joel, L. W., & Philip, S. Y. (1994). *Scheduling parallel tasks to minimize average response time*. Paper presented at the Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms.

Parallel Workloads Archive. from http://www.cs.huji.ac.il/labs/parallel/workload/

Rajkumar Buyya, D. A. J. G. H. S. (2002). Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience, 14*(13-15), 1507-1542.

Sabin, G., Lang, M., & Sadayappan, P. (2007). Moldable Parallel Job Scheduling Using Job Efficiency: An Iterative Approach. In *Job Scheduling Strategies for Parallel Processing* (pp. 94-114).

Srividya, S., Vijay, S., Rajkumar, K., Praveen, H., & Sadayappan, P. (2002). *Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs*. Paper presented at the Proceedings of the 9th International Conference on High Performance Computing.

Sudha, S., Savitha, K., & Sadayappan, P. (2003). *A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs*.

Walfredo, C., & Francine, B. (2000). *Adaptive Selection of Partition Size for Supercomputer Requests*. Paper presented at the Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing.

Walfredo, C., & Francine, B. (2002). Using moldability to improve the performance of supercomputer jobs. *J. Parallel Distrib. Comput., 62*(10), 1571-1601.

Yanmin, Z., Jinsong, H., Yunhao, L., Ni, L. M. A. N. L. M., Chunming Hu, A. C. H., & Jinpeng Huai, A. J. H. (2005). *TruGrid: a self-sustaining trustworthy grid.* Paper presented at the Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on.

Kuo-Chan Huang received his BS and PhD degrees in computer science and information engineering from National Chiao-Tung University, Taiwan, in 1993 and 1998, respectively. He is currently an assistant professor in Computer and Information Science Department at National Taichung University, Taiwan. He is a member of ACM and IEEE Computer Society. His research areas include parallel processing, cluster and grid computing, workflow computing.

Po-Chi Shih received the BS and MS degrees in computer science and information engineering from Tunghai University in 2003 and 2005, respectively. He is now studying PhD degree at computer science in National Tsing Hua University.

Yeh-Ching Chung received a BS degree in information engineering from Chung Yuan Christian University in 1983, and the MS and PhD degrees in computer and information science from Syracuse University in 1988 and 1992, respectively. He joined the Department of Information Engineering at Feng Chia University as an associate professor in 1992 and became a full professor in 1999. From 1998 to 2001, he was the chairman of the department. In 2002, he joined the Department of Computer Science at National Tsing Hua University as a full professor. His research interests include parallel and distributed processing, cluster systems, grid computing, multi-core tool chain design, and multi-core embedded systems. He is a member of the IEEE computer society and ACM.